MacTech Magazine
Vol. 14, No. 1 • January 1998

For Macintosh Programmers & Developers

MacTech®

INSIDE:
The Mac OS, STL & Iterators

Includes Special develop Section by Apple Computer, Inc.

MAC OS 8

CONTEXTUAL MENU BASICS

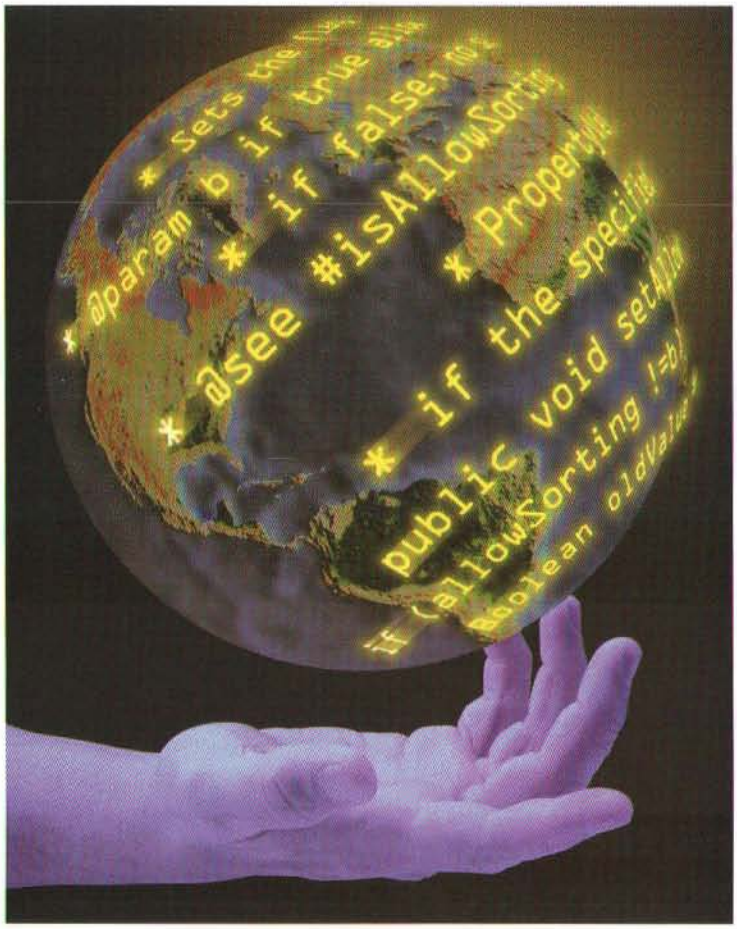
DESIGNING APPEARANCE-SAVVY APPLICATIONS

Plus:

FAST SQUARE ROOT CALCULATION

$5.85 US
$6.95 Canada
ISSN 1067-8360
Printed in U.S.A.

Visual Café™ for Java™: development tools so hot, the competition can't touch them.



	Symantec's Visual Café for Java*	MetroWerks Code Warrior Pro**
Complete RAD Java IDE	Yes	No
Form Designer	Yes	Yes
Display JavaBeans™ in Form Designer	Yes	No
Macintosh runtime for Java (MRJ 2.0 with JDK1.1)	Yes	Yes
Easy conversion from JDK 1.02 to JDK 1.1	Yes	No
Over 100+ JavaBeans with JavaBean Creator Assistant	Yes	No
Interaction Editor	Yes	No
Autocode generation (work the way you want to work)	Yes	No
Database connectivity and support for Filemaker Pro with BlueWorld Lasso	Yes	No
HTML Web authoring tool (Visual Page)	Yes	No
Native support for Oracle, Informix, Sybase MS SQL, & 40+databases via ODBC	Yes	No

*All features included in the Database Development Edition **Features as of October 15,1997

The most powerful Java and JavaBeans environment.

The most powerful Java development solution for databases.

For Macintosh

Download a free trial copy
http://cafe.symantec.com

1-800-971-1582 MT121

The next generation of Java Tools

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
– Joe Zobkiw, author of ***A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

For Macintosh Programmers & Developers

MacTech® MAGAZINE

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 800-MACDEV-1

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@devdepot.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

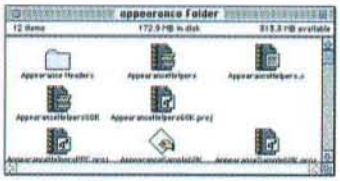
POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

January 1998 • Volume 14, No.1

For Macintosh Programmers & Developers

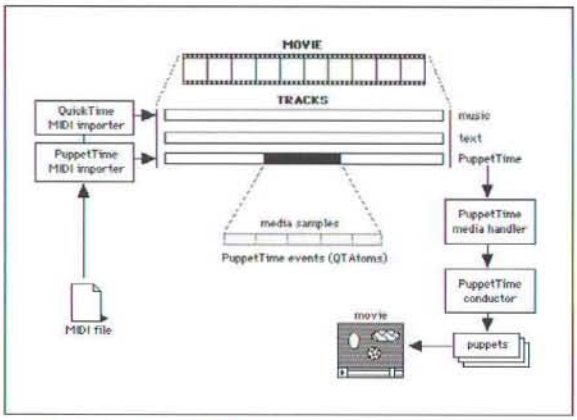
MacTech® MAGAZINE



Designing Appearance-Savvy Applications, page 16

Feature Articles

Columns



Introducing PuppetTime, page 78

Reader Resources

develop Special Section

by Eric Gundrum

A Call for a More Open Mac OS

In recent years, most Macintosh developers have looked to Apple to provide the innovation of the Macintosh platform. We had no choice. To change the look and feel of main components of the operating system meant unreliable, system-wide patches, and these left the developer to the whim of Apple engineers who changed the underlying data structures, on which the developer's patches relied, just because they could. This is a tough area to make a business, but some developers still stuck it out. Nonetheless, there are fewer such products available today, and they generally receive the blame before anything else for unexpected system behavior, rightly or wrongly.

It seems clear that the new Apple no longer has the resources to be as innovative as they once were; we developers have to do more of it ourselves. However, Apple can still help; they can start opening the OS to make it easier for third party developers to replace complete components for the entire system.

Recently, Apple has been developing support for multiple themes to provide the user with choices for the overall look and feel of the Mac interface. I just hope they go far enough. This technology can allow independent developers to produce alternative interface themes. We can have a Windows 95 theme to help the Mac better fit in Wintel environments. We can have a cartoon theme for the kids. Novices can have themes making it easier to learn their way around a computer. Expert users can have themes which allow them to be more productive. Because Apple (hopefully) is clearly defining the API between the various components of themes, we developers can easily explore new interface technologies without the risk of unreliable system patches.

If Apple opens the system enough, and in the right places, we can fix their mistakes (like the gray menus and window backgrounds of the Mac OS 8 Finder) and we can try out new things. In fact, I'd very much like to see a stronger separation between the Finder and the general operation of the operating system. (After all, the Finder is just another application, isn't it?) There are great opportunities to provide alternatives to the Finder. Some people want a Finder alternative that takes less RAM, and they are willing to give up features to get it. I'd like a Finder that makes better use of sound and motion to enhance productivity.

Does anyone remember Sonic Finder and Motion Finder of the late 1980s? Sonic Finder provided audible feedback to various Finder activities such as copying, moving and trashing files. Motion Finder allowed icons and windows to be thrown across the screen. Instead of dragging an icon from the upper left corner of a 20 inch display to the trash in the lower right corner, a simple flick of the wrist was enough to send the icon sailing in the direction of the Trash. If my aim was good enough, the icon made it in, and the file was deleted. There was even a version that combined these features. These tools were developed as explorations in alternative interfaces. Apple probably abandoned them because they would not work for a majority of users. Unfortunately, those of us on the fringe were abandoned in the process.

Now, if only Apple would open up the OS enough that we developers could plug in our own alternative interfaces, then we could again see some real innovation on the Macintosh.

Navigation Services, an Open Design?

One example of Apple moving in the right direction is their Navigation Services technology. This will be a replacement for the Standard File package we have all come to love and hate. Navigation Services will provide a new file system navigation interface for all applications that use it. It is supposed to provide a variety of hooks for developers to enhance its capabilities. I am hoping that with this new technology, Apple also allows developers to write complete replacements of the Navigation Services libraries. Apple is going to the trouble to define the API between Navigation Services and all applications and the API to the file system, Apple also should be open enough to let us developers completely replace the Navigation Services with our own version if we think we have a better idea how to implement it.

There are many similar opportunities for Apple to isolate collections of capabilities into separate libraries, encourage application developers to make use of the new libraries, and allow users to replace those libraries with alternatives written by other developers. (Text Services should be foremost on everyone's mind.) This approach to system software development is similar, in principle, to the component nature of OpenDoc. Rather than have all OS operations dictated by Apple, users could choose to replace Apple's standard behaviors with alternatives that better suit their needs.

Now, I don't advocate that Apple try to turn the Mac OS into another OpenDoc. There are a variety of factors that contributed to the demise of OpenDoc, but there also were some worthwhile features of that technology. We should not throw them all away simply because OpenDoc failed. As long as Apple sticks with their current policy of improving the OS incrementally, they will successfully avoid the single biggest reason for the failure of OpenDoc, QuickDraw GX, PowerTalk and others. Apple has to ease us into using the new technologies one step at a time, and as the hardware advances to support them, that the size of the OS doubles with every release, rather than dump so much on us at once.

Most of all, I want Apple to open the entire OS to third party enhancements, but cleanly. Then I can buy or write my own innovations if I don't like what Apple is supplying. MT

MORE DEVELOPERS PROTECT.

MacHASP® PROTECTS MORE.

Mac™OS

MacHASP Packs More Into Less.

Based on state-of-the-art ASIC technology, MacHASP packs the industry's most advanced security, compatibility and flexibility into a compact and end-user-friendly dongle.

Grow With Aladdin!

The fastest growing company in the industry, with over 4 million keys sold to 20 thousand developers worldwide, Aladdin is setting the standard for software security today.

NSTL Study Rates HASP No.1!

A recent test conducted by the National Software Testing Labs†, the world's foremost independent lab, compared the flagship PC products of leading software protection vendors. The result? HASP was rated the clear overall winner - and number one in all the major comparison categories. For a full copy of the NSTL report, contact your local HASP distributor.

These days, more and more developers are choosing to protect their software against piracy. They're protecting more products, on more platforms, with better protection – and selling more as a result.

And more of these developers are protecting with MacHASP. Why? Because MacHASP offers more security, more reliability and more features than any other product on the market. Only MacHASP offers capabilities such as network support, anti-debugging envelope protection, and secure remote activation and updating.

MacHASP supports the most advanced platforms, including all versions of MacOS and Power Mac – as well as AppleTalk networks. And because Aladdin is a licensed Apple Partner, MacHASP guarantees full, transparent compatibility with the ADB standard.

To learn more about how you can protect better – and sell more – call now to order your MacHASP Developer's Kit.

The MacHASP Developer's Kit contains everything you need to protect your software today!

1-800-223-4277

w w w . a k s . c o m

ALADDIN™

The Professional's Choice

North America	**Aladdin Knowledge Systems Inc.**	Tel: (800) 223 4277, 212-564 5678, Fax: 212-564 3377, E-mail: hasp.sales@us.aks.com
Int'l Office	**Aladdin Knowledge Systems Ltd.**	Tel: +972-3-636 2222, Fax: +972-3-537 5796, E-mail: hasp.sales@aks.com
Germany	**FAST Software Security AG**	Tel: +49 89 89 42 21-37, Fax: +49 89 89 42 21-40, E-mail: info@fast-ag.de
United Kingdom	**Aladdin Knowledge Systems UK Ltd.**	Tel: +44 1753-622266, Fax: +44 1753-622262, E-mail: sales@aldn.co.uk
Japan	**Aladdin Japan Co., Ltd.**	Tel: +81 426-60 7191, Fax: +81 426-60 7194, E-mail: sales@aladdin.co.jp
Benelux	**Aladdin Software Security Benelux B.V.**	Tel: +31 24-641 9777, Fax: +31 24-645 1981, E-mail: 100526.1356@compuserve.com

■ **Aladdin Russia** 095 9230588 ■ **Australia** Contab 03 98985685 ■ **Chile** Micrologica 02 7350041 ■ **China** Shanghai UNI 021 6437 7828 ■ **Czech** Atlas 02 766085 ■ **Denmark** Berendsen 039 577316 ■ **Egypt** Zeineldein 02 3604632 ■ **Finland** ID-Systems 0 8703520 ■ **France** 1 40859885 ■ **Greece** Unibrain 01 6756320 ■ **Hong Kong** Hastings 02 5484629 ■ **India** Solution 011 2148254 ■ **Italy** Partner Data 02 26147380 ■ **Korea** Dae A 02 8484481 ■ **Mexico** SiSoft 91800 55283 ■ **New Zealand** Training 04 5666014 ■ **Poland** Systherm 061 480273 ■ **Portugal** Futurmatica 01 4116269 ■ **Romania** Re Interactive 064 140283 ■ **Singapore** ITR 065 5866788 ■ **South Africa** D Le Roux 011 8864704 ■ **Spain** PC Hardware 03 4493193 ■ **Switzerland** Opag 061 7169222 ■ **Taiwan** Teco 02 5559676 ■ **Turkey** Mikrobeta 0312 4670035 ■ **Yugoslavia** Asys 021 623920

by Dave Mark

Event-Based Programming

How a Mac program communicates with the user

Over the last year or so, we've gotten a lot of feedback about the direction in which you want this column to head. Some folks want more coverage of Java, others PowerPlant and Rhapsody. But the biggest vote of all was a return, for a spell, to the basics. When I first started this column more than six years ago (For you old-timers, my son Daniel, who was born in these pages, is now 5-1/2), we plowed a path for beginners, covering the basics of working with the Mac Toolbox. Since then, we've explored a wide variety of topics and have covered a lot of ground.

Over the next few months, we're going to revisit some of that territory at the behest of a new generation of Mac programmer. We started with last month's column, which gave an updated answer to the question, "How do I get started with Mac programming?" This month, we'll revisit the concept of event-based programming.

Know someone interested in getting started with Mac programming? Hand them your copy of last month's MacTech and point them this way...

Event-Based Programming

Most the programs we've created together have one thing in common. Each performs its main function, then sits there waiting for a mouse click using this piece of code:

```
while ( ! Button() )
 ;
```

This chunk of code represents the only mechanism the user has to communicate with the program. In other words, the only way a user can talk to one of our programs is to click the mouse to make the program disappear! This month's program is going to change all that.

One of the most important parts of the Macintosh Toolbox is the Event Manager. The Event Manager tracks all user actions, translating these actions into a form that's perfect for your program. Each action is packaged into an event record and each event record is placed on the end of the application's event queue.

For example, when the user presses the mouse button, a mouseDown event record is created. The record describes the mouseDown in detail, including such information as the location, in screen coordinates, of the mouse when the click occurred, and the time of the event, in ticks (60ths of a second) since system startup. When the user releases the mouse button, a second event, called a mouseUp event is queued.

If the user presses a key, a keyDown event is queued, providing all kinds of information describing the key that was pressed. An autoKey event is queued when a key is held down longer than a pre-specified autoKey threshold.

Though there are lots of different events, this month we're going to focus on four of them: mouseDown, mouseUp, keyDown, and autoKey. Next month we'll look at some of the others.

Working With Events

Events are the lifeline between your user and your program. They let your program know what your user is up to. Programming with events requires a whole new way of thinking. Up until this point, our programs have been sequential. Initialize the Toolbox, load a WIND resource, show the window, draw in it, wait for a mouse click, then exit.

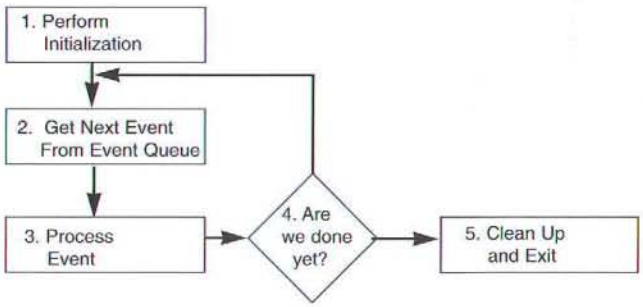
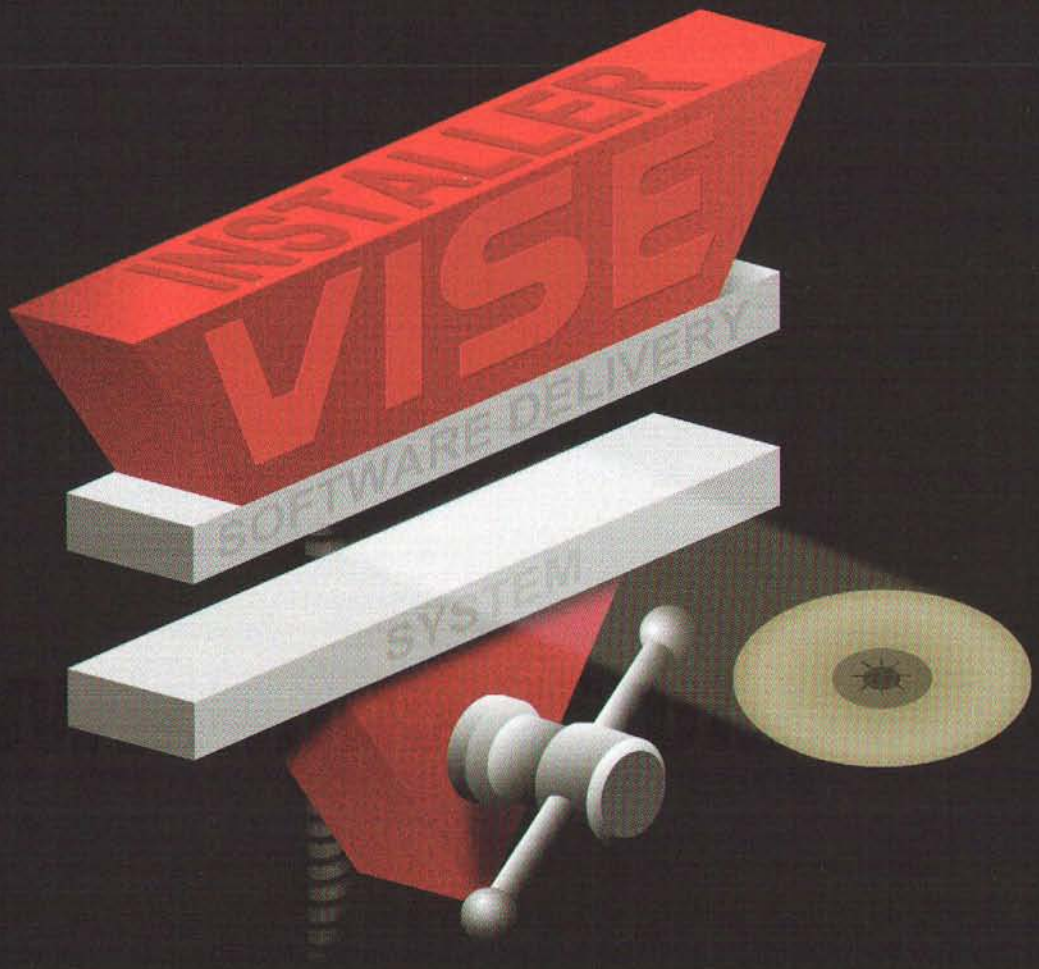


Figure 1. *The main event loop flowchart.*

Event programming follows a more iterative path. Check out the flowchart in **Figure 1**. From now on, our programs will look like this. First, we'll perform our program's initialization. This includes initializing the Toolbox, loading any needed resources, perhaps even opening a window or two. Once initialized, your program will enter the main event loop.

5.0



get it!

www.mindvision.com

The Main Event Loop

In the main event loop, your program uses a Toolbox function named WaitNextEvent() to retrieve the next event from the event queue. Depending on the type of event retrieved, your program will respond accordingly. A mouseDown might be passed to a routine that handles mouse clicks, for example. A keyDown might be passed to a text handling routine. At some point, some event will signal that the program should exit. Typically, it will be a keyDown with the key sequence command-Q, a mouseDown with the mouse on the Quit menu item, or as the result of a Quit event sent by another program like the Finder. If If it's not time to exit the program yet, your program goes back to the top of the event loop and retrieves another event, starting the process all over again.

```
WaitNextEvent() returns an event in the form of an
EventRecord struct:

struct EventRecord
{
  short    what;
  long   message;
  long   when;
  Point    where;
  short    modifiers;
};
```

The what field tells you what kind of event was returned. As I said before, this month we'll only look at mouseDown, mouseUp, keyDown, and autoKey events, though there are lots more. Depending on the value of the what field, the message field contains four bytes of descriptive information. when tells you when the event occurred, and where tells you where the mouse was when the event occurred. Finally, the modifiers field tells you the state of the control, option, command and shift modifier keys when the event occurred.

EventMaster

This month's program, EventMaster, displays four lines, one for each of the events we've covered so far. As EventMaster processes an event, it highlights that line. For example, **Figure 2** shows EventMaster immediately after it processed a mouseDown event.

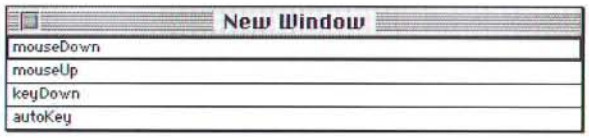


Figure 2. *EventMaster in action.*

EventMaster requires a single resource of type WIND. Create a folder called EventMaster in your Development folder. Next, open ResEdit and create a new resource file named EventMaster.rsrc inside the EventMaster folder.

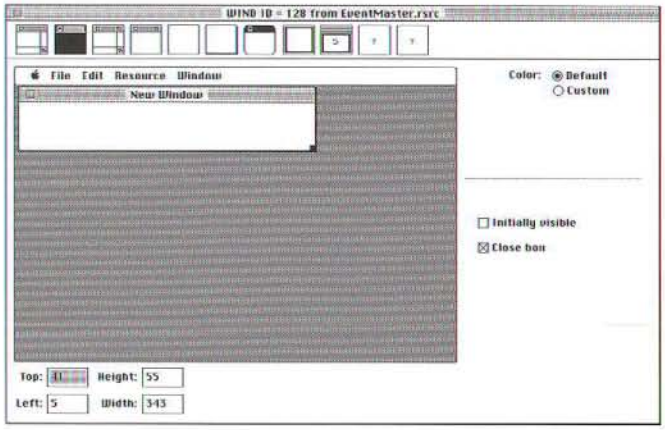


Figure 3. *The WIND resource specifications.*

Create a new WIND resource according to the specs shown in **Figure 3**. Make sure the resource ID is set to 128 and the **Close Box** checkbox is checked. Select **Set 'WIND' Characteristics** from the **WIND** menu and set the window title to EventMaster. Quit ResEdit, saving your changes.

Running EventMaster

Launch CodeWarrior and create a new project based on the MacOS:C/C++:Basic Toolbox 68k stationary. Turn off the **Create Folder** check box. Name the project EventMaster.mcp and place it in your EventMaster folder. Remove SillyBalls.c and SillyBalls.rsrc from the project; we will not be using these files. From the Finder, drag and drop your EventMaster.rsrc file into the project window. You also can remove the ANSI Libraries group from the project, because we won't need them, either. Your project window should look something like **Figure 4**.

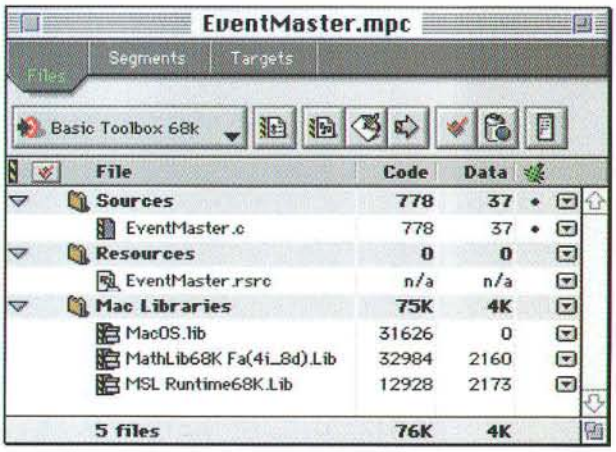


Figure 4. *EventMaster project window.*

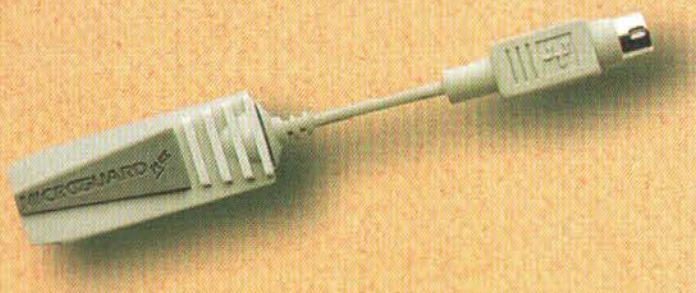
REWARD
$15,000,000,000

Have you seen this man ?
Hacker Harry and the pirate gang are loose and robbing the software stagecoach. if you want to stop them and collect your part of the reward... the place to find your New Generation Protection solution is at:

www.micromacro.com

The expected losses due to piracy in the software industry alone are over $15 billion for 1997. Some of this belongs to YOU. By using MicroGuard's New Generation Copy Protection keys, you will enforce your rights and guarantee your profits.

And now with MicroGuard PC's newest innovative feature, "QUICKGUARD", anyone can secure an entire application within a protective shield in 10 minutes or less. TRY IT.



International
Micro Macro Technologies, Ltd.
3 Hashikma St.
P.O.Box 11516, Azur 58001
Israel

Tel: (972-3) 558-2345
Fax: (972-3) 558-2344
E-mail: info@micromacro.com

USA
MicroGuard
631 South Pontiac St.
Denver, CO 80224
USA

Tel: (303) 320-1628
Fax: (303) 320-1599
E-mail: usa@micromacro.com

SEE YOU AT:

CeBIT HANNOVER

MICROSOFT® WINDOWS™ COMPATIBLE

Mac OS

Charter member Pirate Busters

MICROGUARD™

Select **New** from the **File** menu and type this source code in the window that appears:

```
#include <Sound.h>

#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L
#define kSleep          7

#define kRowHeight      14
#define kFontSize       9

#define kMouseDown      1
#define kMouseUp        2
#define kKeyDown        3
#define kAutoKey        4

/************/
/* Globals */
/************/

Boolean     gDone;
short       gLastEvent = 0;


/************/
/* Functions */
/************/

void ToolBoxInit( void );
void WindowInit( void );
void EventLoop( void );
void DoEvent( EventRecord *eventPtr );
void HandleMouseDown( EventRecord *eventPtr );
void DrawContents( void );
void SelectEvent( short eventType );
void DrawFrame( short eventType );

/************************** main ********/

void main( void )
{
  ToolBoxInit();
  WindowInit();

  EventLoop();
}

/************************** ToolBoxInit */

void ToolBoxInit( void )
{
  InitGraf( &thePort );
  InitFonts();
  InitWindows();
  InitMenus();
  TEInit();
  InitDialogs( nil );
  InitCursor();
}

/************************** WindowInit ********/

void WindowInit( void )
{
  WindowPtr  window;

  window = GetNewWindow( kBaseResID, nil, kMoveToFront );

  if ( window == nil )
  {
    SysBeep( 10 );  /* Couldn't load the WIND resource!!! */
    ExitToShell();
  }

  SetPort( window );
  TextSize( kFontSize );

  ShowWindow( window );
}

/************************** EventLoop ********/

void EventLoop( void )
{
  EventRecord  event;

  gDone = false;
  while ( gDone == false )
  {
    if ( WaitNextEvent( everyEvent, &event, kSleep, nil ) )
      DoEvent( &event );
  }
}

/************************** DoEvent ********/

void DoEvent( EventRecord *eventPtr )
{
  switch ( eventPtr->what )
  {
    case mouseDown:
      SelectEvent( kMouseDown );
      HandleMouseDown( eventPtr );
      break;
    case mouseUp:
      SelectEvent( kMouseUp );
      break;
    case keyDown:
      SelectEvent( kKeyDown );
      break;
    case autoKey:
      SelectEvent( kAutoKey );
      break;
    case updateEvt:
      BeginUpdate( (WindowPtr)eventPtr->message );
      DrawContents();
      EndUpdate( (WindowPtr)eventPtr->message );
  }
}

/************************** HandleMouseDown ********/

void HandleMouseDown( EventRecord *eventPtr )
{
  WindowPtr  window;
  short      thePart;

  thePart = FindWindow( eventPtr->where, &window );

  if ( thePart == inGoAway )
    gDone = true;
}

/************************** DrawContents ********/

void DrawContents( void )
{
  short      i;
  WindowPtr  window;

  window = FrontWindow();

  for ( i=1; i<=3; i++ )
  {
    MoveTo( 0, (kRowHeight * i) - 1 );
    LineTo( window->portRect.right,
        (kRowHeight * i) - 1 );
  }

  MoveTo( 4, 9 );
  DrawString( "\pmouseDown" );

  MoveTo( 4, 9 + kRowHeight );
  DrawString( "\pmouseUp" );

  MoveTo( 4, 9 + kRowHeight*2 );
  DrawString( "\pkeyDown" );

  MoveTo( 4, 9 + kRowHeight*3 );
  DrawString( "\pautoKey" );
```

```
  if ( gLastEvent != 0 )
    DrawFrame( gLastEvent );
}

/********************************** SelectEvent ********/

void SelectEvent( short eventType )
{
  Rect        r;
  WindowPtr   window;

  window = FrontWindow();
  r = window->portRect;

  if ( gLastEvent != 0 )
  {
    ForeColor( whiteColor );
    DrawFrame( gLastEvent );
    ForeColor( blackColor );
  }

  DrawFrame( eventType );

  gLastEvent = eventType;
}


/********************************** DrawFrame ********/

void DrawFrame( short eventType )
{
  Rect        r;
  WindowPtr   window;

  window = FrontWindow();
  r = window->portRect;

  r.top = kRowHeight * (eventType - 1);
  r.bottom = r.top + kRowHeight - 1;

  FrameRect( &r );
}
```

Once the source code is typed in, save the file as EventMaster.c. Select **Add Window** from the **Project** menu to add EventMaster.c to the project. Select **Run** from the **Project** menu to run EventMaster.

When the EventMaster window appears, click the mouse in the window. The mouseDown line will highlight. When you let go of the mouse button, the mouseUp line will highlight. Try this a few times, till you can play the entire drum solo to "Wipeout" on your mouse.

Next, press a key or two on your keyboard (try any key except one of the modifier keys control, option, shift or command). The keyDown line will highlight. Now press the key and hold it down for a while. After a brief delay, the autoKey line will highlight.

Once you're done playing, click the mouse in the EventMaster window's close box to exit the program.

Walking Through the EventMaster Source Code

EventMaster starts off by including <Sounds.h> to access the SysBeep() system call. (SysBeep() used by be part of OSUtils.h, but Apple moved it to Sounds.h as part of Universal Interfaces 3.0.1, inclueded with CodeWarrior Pro 2.) Next we define a series of constants. Some you know, some you don't. The new ones will be explained as they are used in the code.

```
#define kBaseResID      128
#define kMoveToFront    (WindowPtr)-1L
#define kSleep          7

#define kRowHeight      14
#define kFontSize       9

#define kMouseDown      1
#define kMouseUp        2
#define kKeyDown        3
#define kAutoKey        4
```

The global gDone starts off with a value of false. When the mouse is clicked in the window's close box, gDone will be set to true and the program will exit. gLastEvent keeps track of the last event that occurred, taking on a value of either kMouseDown, kMouseUp, kKeyDown, or kAutoKey. We do this so we can erase the old highlighting (if any) before we draw the new highlighting.

```
Boolean     gDone;
short       gLastEvent = 0;
```

As usual, our program includes a function prototype for all our functions.

```
/**************/
/* Functions */
/**************/

void ToolBoxInit( void );
void WindowInit( void );
void EventLoop( void );
void DoEvent( EventRecord *eventPtr );
void HandleMouseDown( EventRecord *eventPtr );
void DrawContents( void );
void SelectEvent( short eventType );
void DrawFrame( short eventType );
```

main() starts by initializing the Toolbox and loading the WIND resource to build the EventMaster window.

```
/********************************** main ********/

void main( void )
{
  ToolBoxInit();
  WindowInit();
```

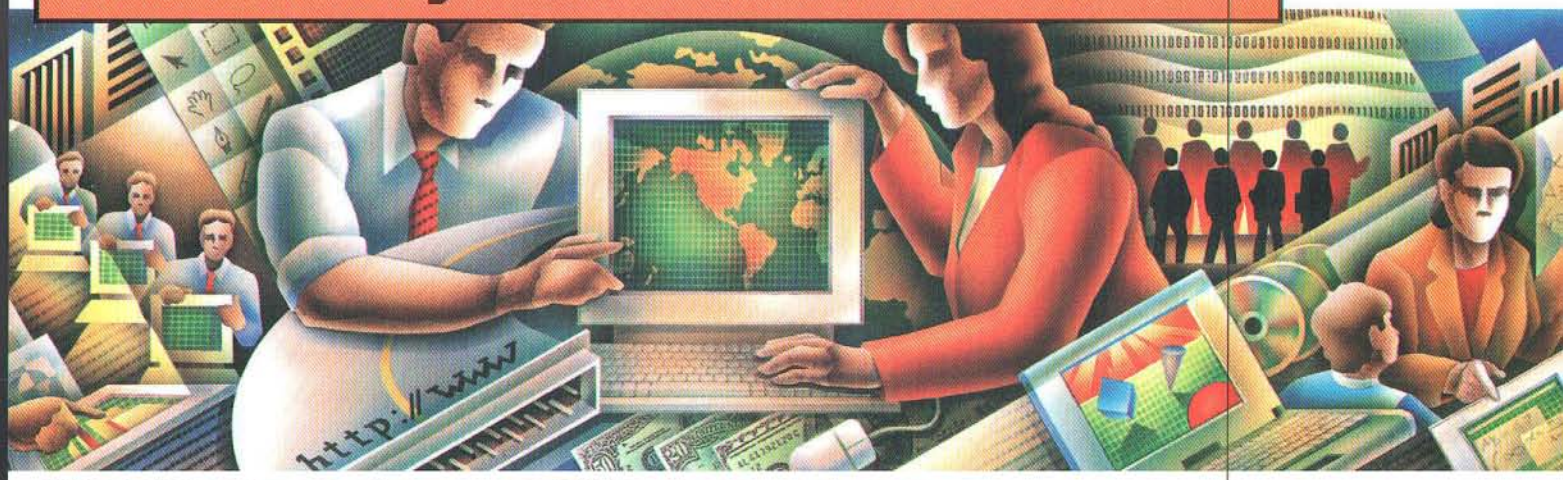
Next, we enter the main event loop.

```
  EventLoop();
}
```

EventLoop() continuously loops on a call to WaitNextEvent(), waiting for something to set gDone to true. The first parameter to WaitNextEvent() tells you what kind of events you are interested in receiving. The constant everyEvent asks the system to send every event it handles.

The second parameter is a pointer to an EventRecord. The third parameter tells the system how friendly your application is to other applications running at the same time. Basically, the number tells the system how many ticks you are willing to wait before recieving the next event, allowing other applications to get some processing time. This number should be about 7 when the application is not busy with a processor intensive task. If the application were doing something where it needed as much processor time as possible (such as compressing a document), this value would be set to zero, but WaitNextEvent() would be

The industry's solution showcase



MACWORLD Expo Conference & Exhibits: January 6-9, 1998

You depend on the Macintosh to get work done faster, to stay connected with colleagues and friends, and turn your best ideas into action. Which is why you can't afford to miss the largest event that brings you the world of the Mac.

Hundreds of New Products and Industry Insights!

Only at MACWORLD Expo can you see and try thousands of products first-hand... learn from Macintosh experts through conferences and keynotes... talk with other Macintosh users and vendors... and stay on top of new developments that could impact your buying decisions. You'll gain valuable insights into how innovative companies are unleashing the power of the Macintosh OS.

Come evaluate cost-saving solutions for:

- Publishing, entertainment and multimedia
- Web site design and Internet navigation
- Networking, intranets and enterprise-wide connectivity
- Education and R&D
- Business and telecommuting

Make plans to attend MACWORLD Expo/San Francisco today!

If the Macintosh is part of your business, MACWORLD Expo is your lifeline. MACWORLD Expo/San Francisco is the industry's premier Macintosh event, and dramatic software introductions scheduled for 1998 may make this year's event the most exciting ever. Gain new insights into the future world of Macintosh... see the hottest and coolest new products... and get the inside view of the Macintosh OS platform. Register to attend today.

Owned & Produced by: IDG EXPO MANAGEMENT COMPANY

Sponsored by: Macworld MacWEEK

For Details:
www.macworldexpo.com
or 800.645.EXPO

Macworld EXPO

san francisco

NEW!

Macworld/Pro Conference

The Macintosh Professionals Conference only at MACWORLD Expo/San Francisco, January 5-7.

Please send more information on MACWORLD Expo

❑ San Francisco/Jan. 1998

❑ Attending ❑ Exhibiting

MT

Name _____

Title _____

Company _____

Address _____

City/State/Zip _____

Phone _____ Fax _____

email _____

Mail to: MHA Event Management, 1400 Providence Highway, P.O. Box 9127, Norwood, MA 02062. Or Fax to: 781-440-0363

THIS IS NOT A REGISTRATION FORM.

called once every seven ticks to give time to the system to check other applications.

The last parameter specifies a home-base region for the mouse. If the mouse moves outside this region, the system will generate a special event, known as a mouse-moved event. Since we won't be handling mouse-moved events, we'll pass nil as this last parameter.

```
/********** EventLoop *********/

void EventLoop( void )
{
  EventRecord   event;

  gDone = false;
  while ( gDone == false )
  {
```

WaitNextEvent() will return true if it successfully retrieved an event from the event queue. In that case, we'll process the event by passing it to DoEvent().

```
    if ( WaitNextEvent( everyEvent, &event, kSleep, nil ) )
      DoEvent( &event );
  }
}
```

WaitNextEvent() is described in detail in *Inside Macintosh: Macintosh Toolbox Essentials*, on page 2-85. If you get a chance, read chapter 2, which describes the Event Manager in detail. You might also want to refer to Chapter 4 in the 2nd edition of the Macintosh C Programming Primer.

`DoEvent()` switches on `eventPtr->what`, sending the appropriate constant to the routine `SelectEvent()`, which highlights the appropriate line in the EventMaster window.

```
/********** DoEvent *********/

void DoEvent( EventRecord *eventPtr )
{
  switch ( eventPtr->what )
  {
```

In the case of a mouseDown, we also pass the event on to our HandleMouseDown() routine, which will check for a mouseDown in the window's close box.

```
    case mouseDown:
      SelectEvent( kMouseDown );
      HandleMouseDown( eventPtr );
      break;
    case mouseUp:
      SelectEvent( kMouseUp );
      break;
    case keyDown:
      SelectEvent( kKeyDown );
      break;
    case autoKey:
      SelectEvent( kAutoKey );
      break;
```

OK, I know I promised we were only going to handle four event types this month, but I couldn't help but sneak this one in here. An update event is generated by the system when the contents of your window need to be redrawn. We'll get to updateEvt next month. In the meantime, if you want to force this code to execute, try triggering your screen dimmer, or cover the EventMaster window with another window and then uncover it..

```
    case updateEvt:
      BeginUpdate( (WindowPtr)eventPtr->message );
      DrawContents();
      EndUpdate( (WindowPtr)eventPtr->message );
  }
}
```

HandleMouseDown() calls FindWindow() to find out in which window, and in which part of the window, the mouse was clicked.

```
/********** HandleMouseDown *********/

void HandleMouseDown( EventRecord *eventPtr )
{
  WindowPtr  window;
  short      thePart;

  thePart = FindWindow( eventPtr->where, &window );
```

If the mouse was clicked in the close box (also known as the goaway box), set gDone to true.

```
  if ( thePart == inGoAway )
    gDone = true;
}
```

DrawContents() draws the contents of the EventMaster window. Notice that the highlighting routine DrawFrame() is only called if a previous event has been handled.

```
/********** DrawContents *********/

void DrawContents( void )
{
  short      i;
  WindowPtr  window;

  window = FrontWindow();

  for ( i=1; i<=3; i++ )
  {
    MoveTo( 0, (kRowHeight * i) - 1 );
    LineTo( window->portRect.right,
        (kRowHeight * i) - 1 );
  }

  MoveTo( 4, 9 );
  DrawString( "\pmouseDown" );

  MoveTo( 4, 9 + kRowHeight );
  DrawString( "\pmouseUp" );

  MoveTo( 4, 9 + kRowHeight*2 );
  DrawString( "\pkeyDown" );

  MoveTo( 4, 9 + kRowHeight*3 );
  DrawString( "\pautoKey" );

  if ( gLastEvent != 0 )
    DrawFrame( gLastEvent );
}
```

SelectEvent() erases the old highlighting (if it existed) and then draws the new highlighting.

```
/********** SelectEvent */

void SelectEvent( short eventType )
{
  Rect       r;
  WindowPtr  window;

  window = FrontWindow();
  r = window->portRect;
```

```
    if ( gLastEvent != 0 )
    {
      ForeColor( whiteColor );
      DrawFrame( gLastEvent );
      ForeColor( blackColor );
    }

    DrawFrame( eventType );

    gLastEvent = eventType;
}
```

DrawFrame() draws the highlighting rectangle.

```
/********** DrawFrame *********/

void DrawFrame( short eventType )
{
  Rect       r;
  WindowPtr  window;

  window = FrontWindow();
  r = window->portRect;

  r.top = kRowHeight * (eventType - 1);
  r.bottom = r.top + kRowHeight - 1;

  FrameRect( &r );
}
```

Some Homework

To understand more about events, read the Event Manager chapters in *Inside Macintosh: Macintosh Toolbox Essentials*. You may have noticed that EventMaster left a lot of room on the right side of each of its event lines. Use this space as a scratch pad, drawing information culled from the EventRecord each time you process an event.

As an example, try writing out the contents of the **when** and **where** fields. How about pulling the character and key codes out of the **message** field of a **keyDown** event. Think of EventMaster as an event playground. Play. Learn.

Next Month

Next month, we'll dig into some events designed specifically for the Window Manager: update and activate events. See you next month... MT

Want to suggest an article for the magazine? Send your suggestion to <mailto:editorial@mactech.com>

by Avi Rappoport

Designing Appearance-savvy Applications

Using Apple's Mac OS 8 Human Interface Guidelines and the Appearance Manager

This was supposed to be a simple review of the *Mac OS 8 Human Interface Guidelines*, a Developer Note from Apple. But, while trying to write the review, I ended up tracking down additional issues, such as what, exactly, the "Mac OS 8 Human Interface" refers to, what "Appearance" means, how the Appearance package updates the Control Manager and the Dialog Manager, and how to go about using this information in your design. Here's what I found.

About the Mac OS 8 Human Interface

Mac OS 8 has some fairly radical changes from System 7 automatically propagated throughout all applications which call the standard Toolbox. These changes include a gray-scale look, windows with draggable borders, button-like popup menus, new controls such as progress bars, an explicit Help menu, and more. All common interface elements are now grouped together under the term ***Appearance***, a concept salvaged from Apple's Copland effort. This Appearance concept will allow future Mac OS releases to support switchable ***themes*** — unified sets of interface elements, including windows, dialogs, controls and color. When themes are supported, designers will be able to create themes without having to program them. Applications which call Toolbox-standard interface elements will automatically display these new interface elements. This will allow users to customize the look of their desktops, without using extensions or changing any basic functionality.

In the meantime, the new ***Appearance Manager*** extends interface elements from the Window, Control, Dialog and Menu Managers. These finally bring Mac Toolbox support for new standard controls, including sliders, progress bars, digital clocks, disclosure triangles and tab panes, as well as menu key command icons and more.

In a striking instance of rationality and clarity, Apple will allow developers to ship this functionality with their applications to run on earlier systems! **Appearance 1.0.1**, an extension/control panel package, is designed to work with versions of Mac OS from 7.1 through 8.0, and runs on 68K and PowerPC processors. You can use these Gestalt selectors to check for the Appearance Manager:

```
/* Gestalt selector and values for the Appearance Manager */

enum {
  gestaltAppearanceAttr         = 'appr',
  gestaltAppearanceExists       = 0,
  gestaltAppearanceCompatMode   = 1
};
```

According to Apple, the Appearance package supports three run-time models: Classic 68K, CFM-68K, and PowerPC CFM. Color QuickDraw is required for Appearance, so it requires a 68020 or better processor. You can download the current Appearance package from Apple's developer site, as described below.

About The Human Interface Guidelines

Back to the design issues and the guidelines: designing the user interface of an application or control panel is not trivial. Designers, and especially programmers working without formal design training, are often confronted with situations where they must present complex information without confusing beginners or annoying expert users. Apple has traditionally provided advice and

Avi Rappoport <avirr@well.com> is interested in user experience design (combining interface and functionality), particularly for information retrieval applications. She is a veteran of StarNine, Metrowerks and Niles & Associates.

Want more from your Mac?

AUTOMATE OFTEN-REPEATED TASKS.

RAPILDLY BUILD APPLICATIONS & PROTOTYPES.

INTEGRATE MULTIPLE APPLICATIONS.

CUSTOMIZE QUARKXPRESS™ & FILEMAKER® PRO.

PERSONALIZE YOUR COMPUTING ENVIRONMENT.

FaceSpan™ 3.0 is a cutting edge interface design and rapid application development (RAD) tool which gives you the power to build and customize scriptable Macintosh applications quickly and easily.

FaceSpan is used in conjunction with AppleScript™ or any other OSA (Open Scripting Architecture) language and takes the place of Visual Basic® on the Macintosh.

Features new to FaceSpan 3.0 give you:

- *MacOS 8 Appearance Manager Support* - Your FaceSpan applications can now include hierarchical menus, tab panels, disclosure triangles, bevel buttons and other MacOS 8 interface objects. FaceSpan 3.0 applications are also compatible with MacOS 8 themes when they become available.

- *Improved Performance* - Your FaceSpan 3.0 applications launch much faster as scripts attached to windows and window items are loaded only when needed.

- *Enhanced Menu Editor* - The FaceSpan menu editor now allows you to setup submenus and assign command key modifiers directly.

"FaceSpan is one of the most wonderful pieces of software I have ever seen."
Howard Oakley, Personal Computer World

"No scripter's workshop should be without FaceSpan."
Tim Warner, 4 Stars, MacWorld

TRANSFORM THE WAY YOU WORK TODAY WITH FACESPAN!

1-800-322-3772 (USA Only)
1-801-226-2984
1-801-226-8438 (Fax)
facespan@dtint.com (E-mail)

http://www.facespan.com

Digital Technology INTERNATIONAL®

design information in the Human Interface Guidelines. This has helped interface designers to be consistent in their standard elements and to create good solutions for situations specific to applications. The *Mac OS 8 Human Interface Guidelines* continue this tradition, describing the changed elements and new features in Mac OS 8, and how to incorporate them in your interface. However, these Guidelines do not include background materials on the theories and principles of interface design: for these, see the earlier *Macintosh Human Interface Guidelines.*

Showing that Apple "gets it" about web distribution, both the original Human Interface Guidelines and this update (among other documentation) are available free on the Web, in both HTML and Adobe Acrobat PDF format at <http://devworld.apple.com/dev/insidemac.shtml>.

The *Mac OS 8 Human Interface Guidelines* is aimed at people designing and implementing interfaces for applications and control panels, so they can make best use of the new features. It includes such specific guidelines as detailed pixel counts of distances between radio buttons, and suggestions for when to use a scrollbar instead of a slider. The writing is clear and concise, and the examples are good — of the high quality we've come to expect from the Inside Macintosh series.

Unfortunately, the **reasons** for changes in Mac OS 8, such as the removal of the border around dialogs, the new text "Help" menu, changes to the scrollbar arrows and the "affordance" (cursor change to indicate move, copy, alias, etc.) are not described at all. Past versions of the interface guidelines explained more of the theory behind design decisions, which educated interface designers and helped them when they had to create elements not covered by the standard interface. In addition, this book does not provide much to help designers use contextual menus or extended menu key commands.

Therefore, this document is only adequate: it provides the basics but does not go beyond. It does not give enough background, theory and examples to ensure that Mac designers create programs with consistent and appropriate use of new interface elements, and that's a shame.

What Is in the Book

The *Mac OS 8 Human Interface Guidelines* covers Controls, Dialog Boxes, Menus, Windows, and Control Panels. I'm going to describe it very briefly, as all interface designers and most programmers should just get a copy and read it themselves!

Old Controls Updated

The Control Guidelines chapter describes the changes to the old controls: push buttons, radio buttons, popup menus, list boxes, scrollbars, edit text fields and static text fields. They have all been updated to a more three-dimensional, "modern" look:

Figure 1. *New Checkbox Control.*

The chapter describes new features, such as "mixed states" for radio buttons and checkboxes (for situations when the selection includes multiple states, such as text with several fonts).

Figure 2. *Mixed State Checkbox (with Disclosure Triangle on the left).*

One of the nice changes is the new disabled states for selected radio buttons and checkboxes, which fixes one of my pet peeves with the classic look. There's no longer a big black bullet or X in the selected control: the selection indicators are now dimmed as well.

In addition, the text of this chapter has been updated using examples from the real world. My favorite:

Avoid the use of negative labels. A checkbox labeled "Delete read messages" with a default state of off is a clearer choice than a checkbox labeled "Do not delete read messages" defaulting to on.

List Boxes now have built-in arrow key support and focus rings, and there are other similar changes that will make all programmers happy.

New Controls

In addition, the Control Guidelines chapter covers the interface to new controls. These are elements that are useful and helpful and were not included in the previous Control Manager.

The controls are:

Figure 3. *Bevel Buttons.*

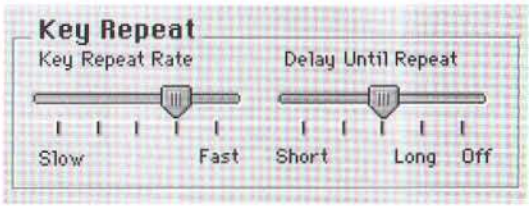


Figure 4. *Horizontal Sliders in a Primary Group Box (there's also a Vertical Slider control).*

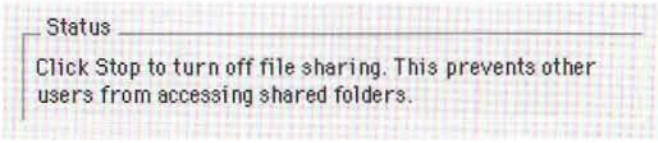


Figure 5. *Secondary Group Box.*

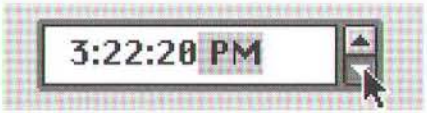


Figure 6. *Digital Clock with Little Arrows.*

Figure 7. *Disclosure Triangle.*

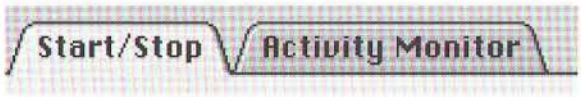


Figure 8. *Tabs (and User Panes to go with them!)*

Figure 9. *Visual Separators (horizontal rule).*

Figure 10. *Placard control, with the new Scrollbar Arrows and Resize Box.*

The new control definition for scroll bars has a variation which supports live scrolling. With a System 7 scroll bar, the user drags a ghost of the scroll box, and the value of the scroll bar changes when the user releases the mouse. With a live scroll bar, the whole scroll box moves, and the value changes as the user drags. You can see this behavior in the Mac OS 8 Finder.

As in the old version, you need a control action procedure to keep up with the changing value of a live scroll bar. But watch out: where the thumb of an old-style scroll bar takes an action procedure with no parameters, a live scroll bar now uses the same action procedure as the other parts of the scroll bar, with parameters.

Figure 11. *Help Icon.*

And there's more than we can show here.

Dialog & Window Appearance

The Alert section of the Guidelines replaces the previous description of alerts, with full explanation of which version to use and how they will appear.

The Dialog Box Guidelines cover such new features as key navigation (using the Tab key to move among edit text fields), and the **focus ring**, now automatically drawn around the field which will accept the keystrokes.

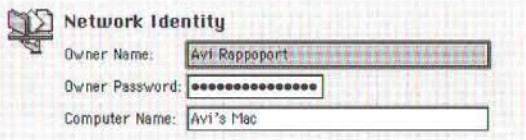


Figure 12. *Focus Ring Around Current Text Field.*

The Layout Guidelines provide complete directions for designing standard spacing of the dialog box or window, text in controls, spacing between controls, and even the Help button. These details will allow interface designers to make clean, elegant layouts without having to guess at the proportions. Use them!

The new standard Utility window is designed for palettes and other non-standard windoids. It can have a top or side title area.

These sections do not describe much of the theory behind the changes in these elements, but they have excellent detail for precise layout.

Menus

The Guidelines finally provide the final word on the location of the Preferences menu item: at the bottom of the Edit menu, after a separator. It will be nice to have the Preferences in the same menu locationin all new applications.

In addition, the Appearance package provides a new standard way to accept and display Control, Shift and Option keys as command-key shortcuts. Use these calls instead of third-party MDEFs for best display on future systems. However, this guide does not define a set of standard meanings for these extended command keys. Losing the opportunity to define these standards is a mistake: developers need rules so users can have cross-application consistency.

Contextual menus are another new feature of Mac OS 8, providing context-sensitive menus when the user holds down the Control key and clicks. This guide offers a good, basic description of how they work, but provides none of the theoretical background or design criteria.

Control Panels

This chapter provides new insights into the issues of Control Panel interfaces. It shows what the standard arrangements should be, and gives guidelines and helpful hints on how to deal with exceptions. New information on fonts, menus, and icons, along with expansion and contractions of a details area should help as well.

Multi-Pane Windows

In the context of Control Panels, the Guidelines discuss the four standard ways of displaying multi-pane windows (using the new and helpful **user pane** toolbox control). They are Tab controls, Buttons, Scrolling list of Icons and Pop-up menus. Unfortunately, my favorite, the disclosure-triangle list used by the CodeWarrior Pro 1 Preferences Window, is not covered. The guide uses four versions of the Appearance control panel to show the strengths and weaknesses of each approach, and provides some suggestions on how to choose between them. This is exactly the kind of practical advice that Apple

should be sharing, although it would be nice if they chose one of these options and used it consistently for the system control panels.

More About Mac OS 8 Changes and the Appearance Manager

The new Guidelines do not cover some of the interface changes in Mac OS 8, and you may be wondering how they came about. Here are some tidbits I picked up at the Developer Conference, and from conversations and discussions on the Apple HI Developers mailing list. These notes apply to the Platinum Appearance only: we won't see the real implications of these changes until the Appearance Manager supports multiple Themes.

Why Modal Dialogs Lost Their Borders

In System 7 and earlier, all dialog boxes have thick gray and black 4-pixel borders, but normal windows have a simple black line border, one pixel on the left and top, two pixels on the bottom and right. This changed in Mac OS 8: now windows have the thick molded border and dialogs are almost borderless.

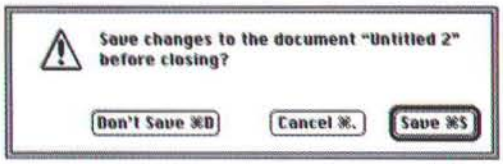


Figure 13. *System 7 Alert showing dialog with border.*

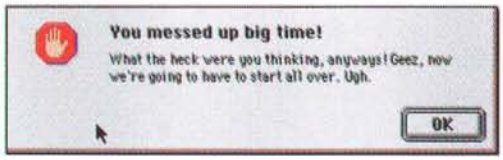


Figure 14. *Mac OS 8 Alert showing borderless dialog.*

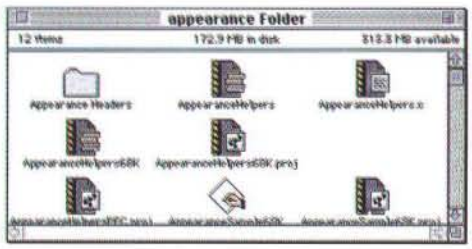


Figure 15. *System 7 Window with thin borders.*

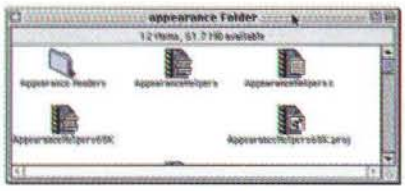


Figure 16. *OS 8 Window showing thick borders.*

The border now signals window draggability — users no longer have to click the title bar to drag, they can drag anywhere. However, users can't drag modal dialogs, so they don't have borders. It's logical but disconcerting, and there's nothing in the new guidelines that explains this.

The Collapse Box

The ***collapse box*** in window title bars shows the WindowShade functionality explicitly.

Figure 17. *Collapse Box.*
(On right side of a window without a close box.)

As most designers have figured out, invisible functions are rarely used or remembered — that's a great part of the Mac's success. So, Apple had to create a way for users to know they could collapse a window; the collapse box is the result. As for why the zoom box moved from the outside right corner to leave room for the collapse box: "*some windows will not have a zoom box but they will have a collapse box. We thought it was more important to keep the consistency between windows with and without [a] zoom box (the collapse box being always at the same place) than between windows of different versions of the operating system.*" [—Arno Gourdol, on the Apple HI Developers mailing list].

Active vs. Inactive States

The new appearance shows more distinction between active elements and inactive ones: "*One of the design goals of the grayscale appearance was to give better definition for active and inactive states. Active objects are beveled and 'pushable' whereas inactive objects are flat, and blend in with the background.*" [—Arlo Rose, on the Apple HI Developers mailing list]

Popup Menu Changes

Rather than a square placard with a subtle shadow and a simple black down-arrow, popup menus now look more like buttons. I like this because it tells the user "push me" rather than "look at me".

Figure 18. *New Popup Menu Appearance.*

In future OS releases, there will be a distinction between pop-up and pull-down menus. "*The new pop-up menu has the dual triangles because when you click on it, the content goes both above and below the menu. Sure this may not mean much now, but in an upcoming release, we plan on introducing a pull-down menu control, and we need to have a visual distinction between the two.*"

"*Pop-ups are for changing state... pull-downs are for executing a command. Since you never want to obstruct the title of a pull-down, the menu will have to be anchored to the control, rather than floating above it.*" [—Arlo Rose, on the Apple HI Developers mailing list]

Scroll Bar Arrow Changes

The scroll bar arrows were changed to up and down pointing triangles from gray arrows for "*aesthetic consistency with the visual appearance of the platinum look.*" [–Arlo Rose, on the Apple HI Developers mailing list]

Help Menu changes

Apple has done a large number of user tests on the Help menu, and finally put the word Help after the last menu on the right for each application. "*Our results would consistently show that people just didn't see it where it was. Most users thought that question mark meant that the clock wasn't sure if the time was right. We tried all combinations of location, and symbol, and the only one people 'got' was the word 'Help' at the end of the application menu list. That and the new sticky menu behavior made for a much higher rate of discovery.*" [—Arlo Rose, on the Apple HI Developers mailing list]

Cursor Affordances

With Mac OS 8, the Finder has a new set of cursors, which provide user feedback on what will happen when they let go of the mouse. For example, an arrow cursor with a + (plus sign) indicates a copy, an arrow cursor with a small right-facing arrow indicates a move, and an arrow cursor with a curved arrow indicates an alias. Although the cursors are included in the Appearance Manager resource file, there are no interface guidelines, or specific functions to enable this behavior automatically within your application.

Writing Appearance-Savvy Apps

To write Appearance-savvy applications, you'll have to avoid nonstandard CDEFs and WDEFs. Most of the old control calls will automatically be routed through the new control manager in the Appearance package, so they will look good even with the new Themes. But any additional controls or window types, such as the Grey Council, will appear as they do in System 7, rather than participating in the systemwide appearance.

The guidelines, however, are not enough. You'll need some of the following tools and sources of information to get started:

- The best material so far on the Appearance Manager is by Edward Voas: "Appearance: Not Just Another Pretty Interface", *develop in MacTech Magazine,* 13:5 (May, 1997), pp. 80-93. It covers both the interface and programming issues, such as the new embedding hierarchy, how to implement live scrolling, and hit testing, and it's incredibly useful! An example application demonstrating the Appearance changes is available on the MacTech website at <ftp://ftp.mactech.com/src/13.05/Appearance.sit>.
- The Mac OS Toolbox Reference covers the toolbox calls for the Appearance Manager. To download, go to <http://devworld.apple.com/dev/insidemac.shtml> and follow the links to the file.
- The Apple HI Developers mailing list provides a chance to ask questions and clarify issues as a group, often with helpful feedback from Apple employees. You can subscribe through the web page at <http://www.lists.apple.com/apple-hi-developers.html>, and archives are at <http://public.lists.apple.com/lists/apple-hi-developers/>.

Appearance Manager SDK

To get the most current Appearance Manager, with the newest features and bug fixes, download it from Apple's DevWorld site at <ftp://devworld.apple.com/MacOS8/>.

Application Frameworks

Metrowerks has announced that the PowerPlant Application Framework version in CodeWarrior Pro 2 supports most of the Appearance features. It wraps both new and old OS calls and directs them automatically according to the presence or absence of the Appearance package, so you can just use the classes and ignore the issue entirely. Please note that PowerPlant has not implemented Contextual Menus in the CodeWarrior Pro 2 release, although there are already third-party shareware classes supporting these new features. Warning: if you've written to the old grayscale classes, you'll have to make some major changes to use the new Appearance classes instead.

Apple's MacApp framework may be updated to support the Appearance package, although the status is not clear right now. MacApp uses the AdLib interface library and licenses some of the Grey Council classes. Current information is at <http://devtools.apple.com/macapp/>.

TCL (Think Class Library) for Symantec C++ seems to be in maintenance mode. I couldn't find any information on support for the Appearance package at <http://www.symantec.com/>.

Resource Editors

ResEdit's templates do not support the new features added by the Appearance Manager, and there is no indication that Apple will ever do so.

Resorcerer version 2 supports the new controls and most of the other Appearance features. It's available from <http://www.mathemaesthetics.com/>.

Constructor for CodeWarrior Pro 2 will support most of the new controls and other Appearance features. For unsupported controls, you can just add an item, enter its attributes, and it will appear as a gray box in Constructor, but will draw correctly when you compile and run.

Conclusion

The Appearance package contains welcome additions to the Mac interface and will promote consistency across applications as well as saving all developers a great deal of effort. The lack of theoretical information in the *Mac OS 8 Human Interface Guidelines* is probably due to the new, leaner Apple. But Apple has been known for it's intelligent, intuitive user experience, and the only way to continue to offer consistency and quality to customers is to work with developers. To keep the edge in interface, as we go forward with the Mac OS and Rhapsody, Apple must invest in both research and communications. MT

by Steve Sheets

Contextual Menu Basics

Adding Support for the new Mac OS 8 Contextual Menu API; even for applications running without Apple's Contextual Menu Manager

A New OS, A New API, A New Logical Interface

The original thought that went into the Macintosh design was "ease of use". This would be the computer for "the rest of us" who were computer novices or worse. After awhile, novices become power users who want quicker ways to do things they already know how to do. You can see this in the menus of the Macintosh. The user has menu items that he can select with the mouse, but he also has Command-key short cuts. You also can see it in drawing palettes that allow simple commands, yet can be double clicked for more in depth configuration.

The *contextual menu* feature introduced with Mac OS 8 continues this idea. While holding down the Control key, a user can click the mouse while above some information he wants to manipulate. The information could be selected text, a picture, some database tables or anything we traditionally think of as exportable data. A popup menu appears to provide him with all the commands related to this data. Even if no data is selected, the user can see commands related to the window, document or application.

Contextual menus provide no new commands that are not already available. Rather it groups the commands in a quick table to be read and selected. Instead of selecting the data and then selecting the command, with one quick click, the user can see what can be accomplish. Searching for familiar commands becomes much easier. (For example, "is Find under Edit, View or Text?") Even if the user knows exactly where the command is in the normal menu, a Control-click on the selected data is faster. Power users will quickly adopt Contextual Menus to get their work done.

Whenever Apple introduced a new "logical interface", they provide a new API to developers so that we can provide this new feature to users. Luckily, the API for contextual menu is very small and involves concepts similar to those in the Drag Manager or even the Scrap Manager. This article explains what a developer must know to simply add contextual menus to his application. As an additional benefit, the article will show how easily Contextual-Menu-like features can be added to an application runing an older version of Mac OS, where Apple has not provided the Contextual Menu Manager.

Extension Modules

There is one exception to the idea that Contextual Menus should not provide commands not listed anywhere else. On a PowerPC Mac with the Contextual Menu API installed, extension modules can add additional functions. These code fragments must be installed in a special folder in the System Folder before startup. They are then loaded and called whenever a contextual menu is created. By looking at the data selected (text, picture, file, folder or even no data), the extensions can add commands to the contextual menu. These plug-ins have only read access to the selected data, so they can not modify the data, but they can return

Steve Sheets has been happily programming the Macintosh since 1983, which makes him older than he wishes, but not as young as he acts. Being a native Californian, his developer wanderings have led him to Northern Virginia. For those interested, his non-computer interests involve his family (wife & 2 daughters), Society for Creative Anachronism (Medieval Reenactment) and Martial Arts (Fencing & Tai Chi). He is currently an independent developer, taking on any and all projects and can be reached by sending mail to <MageSteve@AOL.Com>.

results in the Clipboard or by sending an Apple event. The hosting application need not concern itself with an extension module; all of a plug-in's functions are done without the application's knowledge. The Contextual Menu Extension Module API will be explained in an upcoming article. Until then, just be aware that it is available, but only supported on PowerPC machines.

Adding Commands

The key to determining what to put into a contextual menu is the emphasis — ***contextual*** menus. Ignore that the feature is a popup menu. There are many locations to place commands, or allow short cuts to global settings. The user has selected some content, which he wants to modify. Add commands that are based on the content of what is clicked. In a drawing program, clicking drawing elements (circles, rect, lines) should show the commands that affect the element (rotate, change line thickness, change color). In a word processor, commands that change the style of the text or the arrangement of the document are good choices. Window commands can be available, even if the user clicks a portion of the window that has no data. Think twice before adding an application command. For example, Quit is a poor choice; it already has a short cut in the menu. The API provides a way for sub menus also to be added to the contextual menu. If too many commands need to be added, this is a good way to subdivide the list.

The API

There are only 3 new calls in the Contextual Menu API. In their easiest explanation, the calls initialize the API, check if the contextual menu should appear, and display the Contextual Menu. These functions should be called only if the API is installed on the Mac. Use Gestalt to check if this is the case. The gestaltContextualMenuPresent bit of the gestaltContextualMenuAttr Gestalt attribute indicates if the Contextual Menu Manager is installed.

If the Contextual Menu Manager is installed, call InitContextualMenus(). This will register the application as a client of the Contextual Menu Manager. The routine returns noErr (0) if the initialization succeeds. This check & initialization should occur immediately after the standard toolbox initialization.

Listing 1: ContextualMenuFun.cp
Check if Manager Available, if so Init It

```
long a_result;

if (Gestalt(gestaltContextualMenuAttr, &a_result) == noErr)
  g_have_contextual_menus = BitTst(&a_result,
                           31-gestaltContextualMenuPresent);
else
  g_have_contextual_menus = false;

if (g_have_contextual_menus)
  g_have_contextual_menus = (InitContextualMenus()==noErr);
```

To indicate he wants the contextual menu, the user does a special click on some selected data. For Mac OS 8, this is done by holding down the Control key while doing a standard mouse click. However, there is no reason that in the future this special click could not be done some other way (such as a right mouse click on a two-button mouse). For this reason, the Contextual Menu Manager provides a call that checks if the special click is being done. IsShowContextualMenuClick() should be called after the application is passed any non-NullEvent event. The function is passed the event record and returns true if the special click occured. If the function returns false, the program should continue normally.

Listing 2: ContextualMenuFun.cp
Parse Event, checking if the event is Contextual Menu special click. If so, call routine to handle this.

```
    a_flag = WaitNextEvent(everyEvent, &g_record, 10, NULL);
    if (g_have_contextual_menus) {

// if Contextual Menus event,

      if (IsShowContextualMenuClick(&g_record)) {
// Handle Contextual Menus

        G_ContextualMenu(g_record);
        a_flag = false;
      }
    }

// Handle Normal Events if flag true
```

The most important routine of the Contextual Menu API is ContextualMenuSelect(). It should be called only when IsShowContextualMenuClick() has returned true. Not only does this routine have several parameters, the application must do several preparation tasks before it can be called.

Assuming the application knows which window was chosen, the application should check if this window is the currently selected window. If it is not, it should immediately be made the current selected window, and redrawn. Next, if some content data was selected, some sort of visual feedback should be drawn to indicate this. Inverting, either the data or the outline of the data is a common method to show what was chosen.

Next, a menu handle should be created and inserted into the menu bar just as if the PopUpMenuSelect() call was going to be made. At this point, the various menu items can be added to the menu handle. Sub-menus also can be added. If the Contextual Menu API adds any items to the list (Help or plug-ins), the Contextual Menu Manager will handle parsing these items; your application need not worry about them.

Now call ContextualMenuSelect(). Pass it the created menu handle, the point that was clicked (in global coordinates), a flag that must always be set to false (originally intended to indicate if the clicked area has a Balloon Help item, but now reserved for Apple's use), the Help type needed, the Help string, a pointer to the Apple event descriptor handling the selected data (named inSelection), a pointer to a long integer (resulting outUserSelectionType to explain what happened), and two pointers to short integers (resulting menu ID & menu item).

The Help type parameter can be one of three values: kCMHelpItemNoHelp, kCMHelpItemAppleGuide, kCMHelpItemOtherHelp. If it is kCMHelpItemNoHelp, then there is no help associated with this contextual menu. The Contextual Menu Manager will put the word "Help" in the front of the menu,

but disable it. If the parameter is kCMHelpItemAppleGuide, the Contextual Menu Manager will put the name of the main Apple Guide file into the front of the menu. If the parameter is kCMHelpItemOtherHelp, the Contextual Menu Manager will put the name passed in the Help string into the front of the menu.

If there is no selected data (or if the application does not want to pass the data out), the inSelection parameter can be NULL. Otherwise, it should be a valid Apple event descriptor. After the ContextualMenuSelect() call, the application is responsible for disposing of this descriptor if needed. For simple data structures, the descriptor has a handle to the data (be it text or a picture) as well as the data type. Applications that support scriptability can pass an object specifier pointing to the selected data as the descriptor. This allows some Contextual Menu extensions to send an Apple event back to the current application to change the selected data. Refer to *Inside Mac: Interapplication Communication* for more information about Apple event descriptors and object specifiers.

If the ContextualMenuSelect() call returns successfully, its result is noErr (0). In this case, outUserSelectionType will indicate what happened. If the user fails to select a menu or a plug-in, the constant kCMNothingSelected is returned. If the user selected Show Balloon Help, the Contextual Menu Manager will handle this call, and the constant kShowBalloonSelected is returned (the application should do nothing). If the user selected the Help menu item, kCMShowHelpSelected is returned. The Application should call the appropriate code, Apple Guide or another help system, to display the appropriate help window. If the user selected a menu item, kCMMenuItemSelected is returned, and the menu ID & menu item are also returned.

Before the application parses the result, it should remove any hiliting it placed on the data. The commands the user selected might change this data, so inverting it later may give the wrong results. The application should delete and dispose of the menu handle. Lastly, if the Apple event descriptor was created using Apple event calls, it should be disposed. Listing 3 shows an example of how to setup the basic ContextualMenuSelect() call:

Listing 3: ContextualMenuFun.cp
Sample ContextualMenuSelect call to handle creating & displaying menu.

```
    UInt32 a_type;
    SInt16 a_menu_id;
    UInt16 a_menu_item;
    Boolean a_balloon_available = false;
    Str255 a_help_str = "\pContextualMenuFun Help…";
    UInt32 a_help_type = kCMHelpItemOtherHelp;
    AEDesc a_descr;
    AEDesc* a_descr_ptr = NULL;
// Create Menu
    MenuHandle a_menu_handle = NewMenu(255, "\pX");
    if (a_menu_handle) {
      SetPort(a_window_ptr);
// If Window not front most, bring it front and draw on it
      if (FrontWindow()!=a_window_ptr) {
        SelectWindow(a_window_ptr);
        G_Draw(a_window_ptr);
      }

// If Color Window, hilite picture (invert frame)
      if (a_window_ptr!=g_window_about) {
        SetRect(&g_rect, 0, 0, 200, 200);
        PenMode(srcXor);
        FrameRect(&g_rect);
        PenNormal();
      }

// Insert Menu into Menu list
      InsertMenu(a_menu_handle, -1);
// Fill Item on menu base on Front Window
      G_ContextualMenu_Fill(a_menu_handle, a_window_ptr);
// Fill Descriptor with Pic, if possible
      a_descr_ptr = &a_descr;
      a_descr.descriptorType = typePict;
      a_descr.dataHandle = (Handle) g_pic_1;
// Call to Handle Popup
      OSStatus a_status = ContextualMenuSelect(a_menu_handle,
        p_event_record.where, a_balloon_available,
        a_help_type, a_help_str, a_descr_ptr, &a_type,
        &a_menu_id, &a_menu_item);
// Unhilite
      if (a_window_ptr!=g_window_about) {
        SetRect(&g_rect, 0, 0, 200, 200);
        PenMode(srcXor);
        FrameRect(&g_rect);
        PenNormal();
      }
// If call was ok,
      if (a_status==noErr) {
//   If normal item selected, handle it
        if (a_type==kCMMenuItemSelected) {
          G_ContextualMenu_Action(a_menu_id, a_menu_item,
              a_window_ptr);
        }
//   If help selected, handle it
        else if (a_type==kCMShowHelpSelected) {
          if (g_window_about) {
            ShowWindow(g_window_about);
            SelectWindow(g_window_about);
          }
        }
      }
```

```
// Delete Menu
        DeleteMenu(255);
        DisposeMenu(a_menu_handle);
    }
```

Poor Man's Contextual Menu

Contextual menus were introduced with Mac OS 8. The Contextual Menu Manager also can be installed seperately in any Mac OS 7.6 and later system; although, this is uncommon and is not approved by Apple. While it is always nice to support the latest API, what about the majority of your users who will not be using Mac OS 8 for a time (if ever)? That is where the code G_PoorMan_ContextualMenu provides similar contextual menu functions for these older machines.

G_PoorMan_ContextualMenu uses the PopupMenuSelect() function to create it's own contextual menu. The extension modules will not work with Poor Man's Contextual Menu. Remember, plug-ins are not supported under Mac OS 8 running on a 68040 machine. The code calls the same functions as the normal contextual menu handler, so the developer only has to modify G_ContextualMenu_Fill & G_ContextualMenu_Action for it to work under Contextual Menu Manager API or the G_PoorMan_ContextualMenu code.

G_PoorMan_ContextualMenu must be called whenever the user has clicked the mouse within a window (such as inContent, inDrag, and inGoAway regions). The routine is passed the window the user selected and returns True if this action invokes a Poor Man's Contextual Menu. In that case, the program should not continue to process the event because it has been dealt with.

The routine first checks to make sure that the Contextual Menu API is not installed, and that the current event is a mouse-down in a window. Then it creates a menu handle, to be used by the PopUpMenuSelect() routine. Just like the Contextual Menu API, the Poor Man version has to bringing the window to the top and giving feedback to the selected item. Then the Help item is added to the menu. This is different from the Contextual Menu API, which handles the Help menu. Since this menu may appear on the Help window, the code accounts for this by disabling the Help item in that case. The last bit of setup has G_ContextualMenu_Fill being called for the menu items to be added.

Once the menu is fully created, PopUpMenuSelect() is called to process the popup menu. If an item is selected, the menu item number is examined. If it is one, then the Help menu item was selected, and the Help window should appear. Any other number would cause G_ContextualMenu_Action to be called with the correct number. Finally the temporary popup-menu handle is deleted and disposed.

Listing 4: G_PoorMan_ContextualMenu
Poor Man Contextual Menu (ie. no Manager), Return true if event handled

```
Boolean G_PoorMan_ContextualMenu(WindowPtr p_window_ptr)
{
  Boolean a_flag = false;
// No Manager, event mouse down in Window & Control Key down
  if (!g_have_contextual_menus) {
    if ((g_record.what==mouseDown) && p_window_ptr) {
      if ((g_record.modifiers & controlKey)!=0) {
// Event handled here!
        a_flag = true;
        Str255 a_temp_str = "\p?";
        MenuHandle a_menu_handle = NewMenu(255, a_temp_str);
        if (a_menu_handle) {
          SetPort(p_window_ptr);

// If Window not front most, bring it front and draw on it
          if (FrontWindow()!=p_window_ptr) {
            SelectWindow(p_window_ptr);
            G_Draw(p_window_ptr);
          }

// If Color Window, hilite picture (invert frame)
          if (p_window_ptr!=g_window_about) {
            SetRect(&g_rect, 0, 0, 200, 200);
            PenMode(srcXor);
            FrameRect(&g_rect);
            PenNormal();
          }

// Insert Menu into Menu list
          InsertMenu(a_menu_handle, -1);

// Add About (Dim if About Window on top)
          if (p_window_ptr==g_window_about)
            AppendMenu(a_menu_handle,
              "\p(About ContextualMenuFun…;(-");
          else
            AppendMenu(a_menu_handle,
              "\pAbout ContextualMenuFun…;(-");

// Fill Item on menu base on Front Window
          G_ContextualMenu_Fill(
              a_menu_handle, p_window_ptr);
// Call to Handle Popup
          long a_result = PopUpMenuSelect(a_menu_handle,
            g_record.where.v, g_record.where.h, 1);
// Unhilite
          if (p_window_ptr!=g_window_about) {
            SetRect(&g_rect, 0, 0, 200, 200);
            PenMode(srcXor);
            FrameRect(&g_rect);
            PenNormal();
          }
// If call was ok,
          if (a_result) {
            short a_menu_id = HiWord(a_result);
            short a_menu_item = LoWord(a_result);
            if (a_menu_id==255) {
//    If help selected, handle it
              if (a_menu_item==1) {
                if (g_window_about) {
                  ShowWindow(g_window_about);
                  SelectWindow(g_window_about);
                }
              }
              else {
//    If normal item selected, handle it (Adjust for Help Menu Item)
                a_menu_item -= 2;
                G_ContextualMenu_Action(a_menu_id,
                  a_menu_item, p_window_ptr);
              }
            }
          }
// Delete Menu
          DeleteMenu(255);
          DisposeMenu(a_menu_handle);
        }
      }
    }
  }
  return a_flag;
}
```

Sample Code

ContextualMenuFun was created to show how easy it is to provide contextual menus to the user. Most Macintosh developers will recognize their way around the standard portions of the program. All the code to handle details of a Contextual Menu is inside of the G_ContextualMenu routine. The Non-Contextual Menu Manager version of the code is inside G_PoorMan_ContextualMenu. Both routines call G_ContextualMenu_Fill & G_ContextualMenu_Action. G_ContextualMenu_Fill is passed the menu handle & selected window pointer, and fills the menu with items. Assuming the user has chosen a command, G_ContextualMenu_Action is passed the clicked window, the selected menu ID & menu item. G_ContextualMenu_Action parses this information, and takes action. The code was written to be expanded, which is why menu ID is passed. Remember, sub menus can be added so the menu ID is not always the same. The selected window is passed so commands can be more window and content sensitive.

The new "logical interface" begs for an object oriented approach to handling the contextual menus. A clicked visual object calls the standard routine to set up the contextual menu, using the API or not, which calls a virtual method to fill in the menu handle with commands. The standard routine would then display the menu. Assuming the user selects an item, another virtual method would be called to take action. Controls and field objects could even put their commands into the menu and then call the owning window for the window to put in its own commands, and so on up the command chain. Commercial or in-house frameworks easily could be modified this way to support the Contextual Menu API. MT

by Nat McCully, Senior Software Engineer, Claris Corporation

Supporting Multi-byte Text in Your Application

How to make most efficient use of the Script Manager and International APIs in your text engine

Introduction

You have developed the next greatest widget or application and you want to distribute it on the Net. Your application has a text engine, maybe simple TextEdit, or one of the more sophisticated engines available for license, or even one you wrote yourself. One day, you get an e-mail from someone in Japan:

> Dear Mr. McCully,
> My name is Takeshi Yamamoto and I use your program, MagicBook, everyday. But, I have a problem using Japanese characters in it. When I hit the delete key, I get weird garbage characters. My friends and I wish to use both Japanese and English in your program, but it does not work properly. Please fix it!
> T. Yamamoto

Suddenly, there are people on the other side of the world who want to use your application in their language, and you are faced with a dilemma. You have no first-hand knowledge of the language itself, but you may be somewhat familiar with the Macintosh's ability to handle multiple languages in a single document with ease. Simply cracking open *Inside Macintosh: Text* seems daunting. How will these new routines affect the performance of your program? Will you introduce unwanted instability and anger your existing user base? Where can you find information on how to use which routines best, not just a description of what each routine does?

This article attempts to address some of these issues, and in general familiarize you with some of the best things that make the Mac an excellent international computing environment. Intelligent use of the Macintosh's international routines, WorldScript, and the other managers in the Toolbox can be the difference between a US-only application and a truly "world-ready" tool that any user, anywhere, can utilize as soon as they download it to their hard disk. Although this paper deals primarily with Japanese language issues, the concepts outlined herein can be used with any multi-lingual environment.

What is WorldScript?

WorldScript is the set of patches to the system that enables the correct display and measurement of multi-lingual text. Over time, many of these patches have been rolled in to the base system software, but even in MacOS 7.6.1, you will find a set of WorldScript extensions in the Extensions folder when you install one of the Language Kits available from Apple. The concepts and code snippets in this paper will work equally well on, for example, the Japanese localized Mac OS, or on a standard U.S. system with the Japanese Language Kit (JLK). A good source of localized system software and Language Kits is the Apple Developer Mailing CD-ROM, available from the Apple Developer Catalog. WorldScript is one of the Apple-only technologies that makes multi-lingual computing possible in a far easier way than the other guys. When it comes to having Chinese, Korean and Japanese all in the same document, WorldScript, on Mac OS, is the only thing out there.

Nat McCully has been at Claris in the Japanese Development Group for the last 6 years. He has worked on numerous Japanese products, including MacWrite II-J, Filemaker Pro-J, Claris Impact-J, ClarisDraw-J, and ClarisWorks-J. He speaks, reads and writes Japanese, and enjoys traveling in Japan. He is currently working as Development Lead on the next release of ClarisWorks-J.

Multi-byte Text on the Mac

OK, let's get to the meat, you say. How do you make your text engine handle two-byte characters? Well, before giving you a bunch of code, let's explain how the Mac handles two-byte text.

What is a Script?

Each language that the Mac supports is grouped into categories called "scripts." For example, English and the other Roman letter-based languages like French and German all belong to the Roman script. Japanese belongs to the Japanese script. Character glyphs in the Roman script are each represented by a single-byte character code. Japanese characters are represented by a 16-bit (2 byte) character code.

Setting Up the Port: Pre-System 7 API's versus New API's

On the Mac, each font is also associated with a script. There are Roman script fonts like Helvetica and Palatino, and Japanese script fonts like Osaka and Heisei Minchou. As you know, when text is drawn into a QuickDraw grafPort, you first set up the port with the appropriate font, size and style, and then call DrawText() to draw the text. If you are using the old Script Manager API's (like CharByte() and GetEnvirons()), you need to set the port to a font in the script you are interested in processing. Once the port is set to a particular font, calls to the Script Manager will follow the rules of that font's script. So the port, by way of setting the font, also has an implicit script setting. This is the key to using the Script Manager routines so they will return the correct information to your application using the older API's. The new API's have a script parameter, so it is not necessary to set the font of the port before using them. Since the Script Manager doesn't have to call FontScript() to find out the script of the current font before passing the script to WorldScript, using the newer API's could speed up your application is certain cases.

Adding Script-savvy Features to your Application

First, you need to determine if the user's system has a non-Roman script system installed. Many programs boost performance by calling Script Manager routines only when absolutely necessary. One way to find out all the scripts installed is to loop through all 33 possible script codes (smRoman being 0 and smUninterp being 32) and calling GetScriptManagerVariable() with the selector smEnabled. Roman script is always enabled.

Listing 1: Finding Which Scripts are Installed

```
ScriptCode script;
for (script = smRoman + 1; script <= smUninterp; script++)
{
  if (GetScriptManagerVariable(script, smEnabled))
    InitInternalScriptInfo(script);
        // non-Roman script present...
}
```

InitInternalScriptInfo() refers to a routine you write that will initialize your internal data structures that deal with specific script systems, such as line-breaking tables, on a script-by-script basis.

Line Breaking

Most applications don't rely entirely on the Script Manager for line breaking, hit testing, or word selection, because using those routines is thought to be too slow. It is possible to optimize your text engine so that you incorporate the correct behaviors for each script system present, while maintaining the highest possible performance. The Toolbox call for finding line breaks is StyledLineBreak(). To use it, however, you must restrict the text you pass it to lengths of less than 32K (actually, this is true of the whole Script Manager, so tough) and text widths to whole pixel values (can you say 'rounding error?'), and if you are explicitly scaling the text, it won't work at all. You must also organize the text you pass to it in terms of script runs and style runs within them. Therefore, most applications that have word-processing functionality choose to implement their own line-breaking code that is customized for their own needs. Unfortunately, many of these private implementations break when used on WorldScript systems.

Line Breaking with Japanese Text

The simplest line-breaking algorithm for English text is to look for a space (ASCII 0x20) character in the line near the graphic break, and if there is none, to break on the byte-boundary nearest the graphic break. Japanese text is a bit more complicated. Japanese text has no spaces, so you must break at the character boundary nearest the graphic break. There is an additional wrinkle: Certain characters are not

allowed to begin a line, and certain characters are not allowed to end a line. This set of line-breaking rules is referred to as *Kinsoku shori*. For example, you cannot begin a line with a two-byte period. You cannot end a line with a two-byte open parenthesis followed by more text on the next line. A list of *kinsoku* characters is available from the Japanese Standards Association in the form of a Japanese Industrial Standard (JIS) document. It is also in Ken Lunde's excellent book, *Understanding Japanese Information Processing*, in the section entitled "Japanese Hyphenation." While not all Japanese agree on the correct set of *kinsoku* characters, this set is a good default. Some applications allow the user to edit the *kinsoku* character set to their own liking.

Once you know that the current byte offset in your text is on or just before the graphic break, you need to see if that byte is part of a two-byte character. Then you need to see if the character is a character that can't end a line. Then you need to check the character after it to see if it is a character that can't begin a line. This can be repeated as necessary, for support of a string of *kinsoku* characters. For example, suppose the character on the graphic break (the break char) can end a line, but the character after it can't begin a line, causing the break char to wrap. However, the character before the break char is one that can't end a line, so you must then check the char before it, and so on, and so on, and... The example below is simplified to illustrate a particular case; actual code for an application would probably be organized differently.

Listing 2: Checking Graphic Break Char with Kinsoku Processing

CheckGraphicBreak

Check a text run's length against the pixel edge of the line (the so-called graphic break), and then adjust the line break if there is an illegal *Kinsoku* character on the break.

```
UInt16 * gStartLineKinsokuChars;  // chars that can't begin
                                  // a line.
UInt8    gNumStartKinsokuChars;   // number of chars above.
UInt16 * gEndLineKinsokuChars;    // chars that can't end a line
UInt8    gNumEndKinsokuChars;     // number of chars above.

// This function will return FALSE if the char at offset
// is not a valid break point. It checks the char after it,
// but not the char before it, for kinsoku.
static Boolean CheckGraphicBreak(UInt8 *     textPtr,
                                 UInt16   offset,
                                 ScriptCode script);
{
  SInt16 result;

  // The textPtr starts at a known 'good' character
  // boundary. In this case it is the beginning of the
  // line, but it could be the beginning of the
  // stylerun.

  // Find out if script only has 1-byte chars. If so,
  // we assume it's ok to break at this char.
  if (GetScriptVariable(script, smScriptFlags) &
          (1 << smsfSingByte))
    return TRUE;
```

```
    result = CharacterByteType((Ptr)textPtr,
              offset,
              script);
    if (result == smSingleByte)
      return TRUE;  // In real life, you're not done
                    // until you check the chars before
                    // and after this one for kinsoku.
    if (result == smFirstByte)
      return FALSE;
    if (result == smLastByte)
    {
      UInt8    index;
      UInt16 theChar = *(UInt16 *)&textPtr[offset - 1];

      // Now we have a valid break on a 2-byte char.
      // We need to check if it's a kinsoku character.
      // This code checks Japanese kinsoku only, but
      // with a little work this could be extended to
      // all 2-byte scripts that don't break on spaces.
      if (script != smJapanese)
        return TRUE;

      for (index = 0; index < gNumEndKinsokuChars; index++)
      {
        if (theChar == gNumEndKinsokuChars[index])
          return FALSE;
      }

      // Now we check the char after this one, in case it
      // is a char that can't start a line. First see if
      // it's a 1-byte char. In real life, there are 1-byte
      // kinsoku chars to check for.
      if (textPtr[offset + 1] == NULL ||
        CharacterByteType((Ptr)&textPtr[offset + 1],
              0, script) == smSingleByte)
        return TRUE;

      theChar = *(UInt16 *)&textPtr[offset + 1];

      for (index = 0; index < gNumStartKinsokuChars;
                index++)
      {
        if (theChar == gNumStartKinsokuChars[index])
          return FALSE;
      }
    }
    return TRUE;
}
```

Hit Testing

Hit testing is another area in your text engine that demands the highest possible performance. When the user clicks in the text, any delay in setting the insertion point there will be noticed. Drag selection is another example of the same code working hard to find the character boundaries and setting the correct hilite area.

Some applications use a locally allocated cache of possible first byte character codes that they use to test a particular character in the text stream for byteness. This is simple to create with the Toolbox call FillParseTable(). FillParseTable() returns in your pre-allocated 256 byte buffer all the bytes that can be a first byte of a two-byte character in the script you pass to it. Be aware that in some scripts, some character codes can be both the first byte of a two-byte character as well as the second byte of a two-byte character, depending on their context within the text stream. Therefore, you need more than just this information to successfully find out what kind of character the byte you're interested in is a part of. In a mixed stream of text with both one-byte and two-byte characters, using the parse table in a single pass over the text is much faster than calling CharacterByteType() for each byte. An example of this is below, in a sample function that goes through a text stream and counts the number of characters in it:

Listing 3: Counting the Chars in Mixed-byte Text

CountCharsInScriptRun

Counts the number of characters in a run of contiguous script (font), checking for both one-byte and two-byte characters.

```
UInt32 CountCharsInScriptRun(UInt8 * textPtr, UInt32 length,
          ScriptCode script)
{
  UInt8    parseTable[256];
  UInt32 curByte, charCount;


  (void)FillParseTable(&parseTable, script);

  for (curByte = 0L, charCount = 0L; curByte < length;
              curByte++)
  {
    if (parseTable[textPtr[curByte]] == 1)
      continue;
    charCount++;
  }
  return (charCount);
}
```

Notice that because we started at a known 'good' boundary, we were able to test only the first bytes of the two-byte characters in the stream as we counted along. This code would not work in all cases if we started at an arbitrary point in unknown text, because of the ambiguity of the byteness of some character codes in some scripts. Caching the parse tables for all installed scripts in the user's system at launch time would further speed up your processing, so you wouldn't have to call FillParseTable() every time.

Measuring Two-byte Characters

On the Mac, all two-byte characters are the same width. In a future system software release, proportional two-byte characters will be supported, but up until now all two-byte-savvy applications assume mono-spaced two-byte characters, and even if proportional characters are supported, they will be mono-spaced by default so as not to break every application currently shipping.

Before the Mac OS supported measuring two-byte characters with TextWidth(), a special code point in the single-byte 256 char width table was reserved for the two-byte character width for that font. In the Japanese and both Chinese scripts, this code point is 0x83. In Korean script, it is 0x85. This code point still works, even though Apple now recommends you use TextWidth() for all measuring of multi-byte or mixed text. In the future for proportional measuring, TextWidth() will probably be what you will use.

Below is an example of a function that measures any text, and returns the amount in a Fixed variable. This is useful if you are measuring text and the user has Fractional Glyph Widths turned on (meaning you made a call to SetFractEnable()).

Listing 4: Measuring Mixed-byte Text

GetTextWidth

This function supports multiple styleruns in the text, and consists of two loops: One for the styles and one for the characters in each style. The port is set on each stylerun and then Macintosh APIs are called to measure the text.

```
#define JSCTCWidthChar 0x83
#define KWidthChar  0x85

typedef struct tagStyleRun {
  UInt32   styleStart;
  UInt16   font;
  UInt16   size;
  UInt8    face;
} StyleRun, *StyleRunPtr;

Fixed GetTextWidth(UInt8 * textPtr, UInt32 length,
                   StyleRunPtr styleRuns, UInt32 numStyles)
{
  ScriptCode      curScript;
  UInt32          byteNum, styleNum;
  Fixed           totalWidth = 0L;
  ScriptCode      curScript;
  FMetricRec      curFontMetrics;
  WidthTable **curWidthTable;
  UInt8           parseTable[256];

  // loop thru each stylerun, measure its characters
  for (styleNum = 0L; styleNum < numStyles; styleNum++)
  {
    // Set up the port (in real life, you'd restore the
    // old settings when you exit)
    TextFont(styleRuns[styleNum].font);
    TextFace(styleRuns[styleNum].face);
    TextSize(styleRuns[styleNum].size);
    FontMetrics(&curFontMetrics);
    curWidthTable = curFontMetrics.wTabHandle;
    HLock((Handle)curWidthTable);
    curScript = FontScript();
    (void)FillParseTable(&parseTable, curScript);

    // loop thru each char in the stylerun
    for (byteNum = styleRuns[styleNum].styleStart;
        (styleNum + 1 < numStyles &&
          byteNum < styleRuns[styleNum+1].styleStart) ||
        (styleNum + 1 >= numStyles && byteNum < length);
          byteNum++)
    {
      if (parseTable[textPtr[byteNum]] == 1)
      {
        if (curScript == smJapanese ||
            curScript == smTradChinese ||
            curScript == smSimpChinese)
          totWidth +=
            (*curWidthTable->tabData)[JSCTCWidthChar];
        else if (curScript == smKorean)
          totWidth +=
            (*curWidthTable->tabData)[KWidthChar];
        else
          totWidth += (Fixed)
            TextWidth(&textPtr[byteNum], 0, 2) << 16;

        byteNum++;
      }
      else
        totWidth +=
          (*curWidthTable->tabData)[textPtr[byteNum]];
    }
    HUnlock((Handle)curWidthTable);
  }

  return (totWidth);
}
```

The above function still makes expensive calls like FontMetrics(), FillParseTable() and TextWidth() on each stylerun. It would be an even better idea to have a local cache of the width tables and parse tables of fonts you know are in the document, so you don't have to rebuild them every time the user clicks or drags or types in the text.

So, now that you have a relatively fast way of measuring the text, you can use it to find the pixel value of any character in the text, and use that for your internal CharToPixel and PixelToChar logic. Or, you can use the Mac OS Toolbox calls CharToPixel() and PixelToChar(), which will always work on any script but may be slower.

Localizing Your Application for Japan

Now that we have reviewed a few of the basic text engine issues for handling two-byte text, there are a few things about Japan in particular that make localization a challenge.

Japan is possibly the most interesting major software market to localize for, if you are interested in text and text layout. It is a mature market, with a diverse number of products enjoying many millions of dollars in sales each year. The Macintosh has a larger market share there than in the U.S. or Europe. Text in Japan has traditionally been difficult to input and output using machines, and the use of text in graphic design requires that the text layout be extremely flexible. The characters are complex (so complex that bolding them may make them illegible), and emphasis or adornment has forms that use background shading, different types of lines around the text, and even dots or ticks above or to one side of each character. Condensed and extended faces have different results on PostScript printers than they do on QuickDraw. Bold and italic faces were not supported on the first PostScript Japanese printers. Underlines are not drawn by QuickDraw when the font is a Japanese font. These last two things might be fixed in future releases of the system software, but for now the application developer must work around them.

For underline, you must draw a line under the text. The reason QuickDraw doesn't draw it for you is that it usually uses the font's baseline as the underline location, but Japanese fonts' two-byte glyphs take up more room and descend below the baseline. Where you draw your underline is up to you, but take a look at how other Japanese programs do it and make it fairly consistent.

Vertical text is pretty much a checkbox item nowadays in Japanese word-processing programs. Most novels and many magazines are layed out vertically, but until recently computers were horizontal-only. While the Windows95® APIs support drawing text vertically, the Mac OS still does not, outside of using QuickDrawGX typography (which is excellent, by the way). In comparing vertical text to horizontal text, several things change about the line layout: The first line starts at the top right, and the text flows down to line-end, then wraps to the next line, which is to the left of the first line; the baseline is generally considered to be in the center of the line; underlines are drawn to the right of the text, as are emphasis dots; two-byte characters are not rotated, but single-byte characters are, 90° clockwise; certain characters have vertical text variants, like many punctuation characters. Where these variants are in the font can be found in the 'tate' table in the font ("tahteh" means "vertical" in Japanese).

Rubi are small annotation characters, placed above, below or to the side of the text they annotate. Usually they provide pronunciation guidance for unusual or hard-to-pronounce Kanji characters.

Date formats in Japan include the current year of the emperor's reign; again, supported on Windows95® but not on Mac OS. It is up to the application to support these formats if so desired. Also, date formats 2 and 3 produce identical results, due to the fact that the abbreviated month and the long month are the same thing in Japanese. Japanese applications may opt to substitute a different format in one of those formats' place.

Find and Replace needs to be expanded to include the different types of characters used in Japanese. Standard Japanese text may contain any of the following types of characters: one-byte Roman, two-byte Roman, one-byte numerals and symbols, two-byte numerals and symbols, one-byte katakana syllables, two-byte katakana syllables, two-byte hiragana syllables, and two-byte Kanji characters. The hiragana and katakana characters are equivalent in terms of the sounds they represent in Japanese, so a good Find/Replace function should include an option to find the search string in either syllabary.

Sorting in Japanese is difficult because the Kanji characters can have different pronunciations depending on their context. To sort Kanji correctly, you need a separate kana key field that indicates the pronunciation and you sort on that. Also, Mac OS CompareText() doesn't sort the long sound symbol correctly (that symbol changes sound depending on the character before it, but Mac OS always sorts it in the symbols area), so for linguistically correct sorting you need to write your own sorting routine.

If your application supports character tracking using the Color QuickDraw function CharExtra(), be aware that the CGrafPort member chExtra only uses 4 bits for signed integer values and the other 12 bits for the fraction. The value you pass to CharExtra() is a Fixed value of how many pixels you want to track out (or in) the text, and QuickDraw divides that by the current text size, to arrive at the chExtra value. This means that if the tracking value you pass to CharExtra() is greater than 8 times the text size, the chExtra field will go negative, and your text will be drawn incorrectly. Unfortunately, Japanese text is routinely tracked out beyond this limit in many applications. The only workaround is for you to draw the text one character at a time, and use the QuickDraw pen movement calls like MoveTo() to move the pen yourself. The same is true for SpaceExtra().

Inline Input

Inline input of Japanese, Chinese or Korean is a way of using an intermediate program (called an Input Method) to translate your keystrokes into the many thousands of possible characters in those languages, all in the same place on screen that you would normally see characters typed in the line. In Japanese, the Input Method changes your keystrokes into

phonetic Japanese kana characters, then converts some of those characters into Kanji characters to form a mixed kana and Kanji sentence. Then the user hits the return key to confirm the text in the line, ending the inline input session. Inline input on the Mac on System 7.1 or later uses the Text Services Manager (TSM). If your application uses TextEdit as its main text engine, you can support inline input quite easily using TSMTE. If you have your own text engine, you will need to do more work to support TSM Inline Input.

TSM uses Apple events to send and receive data between your application and the Input Method. You must implement several AppleEvent handlers, the most complex of which is the kUpdateActiveInputArea. In that handler, you must draw the text in all its intermediate stages, as the user is composing and editing the Japanese sentence before s/he confirms it to the document. If there is text after the so-called 'inline hole,' you must actively reflow the text if such editing causes the length to change. Each time the user makes a change, the text in the inline hole is received from the Input Method in an Apple event. The application draws it in the text stream, along with special Inline styles that help the user tell which text in the inline hole is raw (unconverted) text, which is converted text, which is the active phrase, where the phrase boundaries are in the inline hole, and other information.

After implementing the TSM support in your application, it is imperative that you test it with third-party Input Methods. At the time TSM was introduced, the documentation for how to write an Input Method was still a little spotty. This resulted in each Input Method handling text slightly differently. Also, Kotoeri, Apple's Input Method, has fewer features than the leading third-party Input Methods. Be sure to test your application with all of them you can find, so you can verify that it won't crash or produce strange results. Some Input Methods have strange quirks, like always eating mouseDown events, or having different requirements about how large a buffer they can handle without crashing. This knowledge comes from testing, and sometimes can be found on the Internet in Usenet newsgroups (in Japanese).

What About Unicode?

Unicode is being billed as the latest panacea for the problems of internationalization. What does Unicode give you? Where does it fall short?

Unicode was designed to solve one problem: There are many incompatible, overlapping encoding schemes for different languages, and supporting all of these encodings is a complex problem. What if there was a single encoding scheme that supported all the writing systems of the world, and guaranteed that you could display text in all the languages Unicode supports if only you had the right Unicode font for each language? Unicode tries to be that encoding.

For Japanese text data, the Mac OS and Windows95 use Shift-JIS internally, while Rhapsody and WindowsNT® use Unicode. On the Internet, most Japanese text is encoded using the 7-bit ISO-2022-JP standard. Whether or not you use Unicode to represent text internally to your application, you will have to support all three standards for full file and data compatibility with the rest of the world. In Unicode, all characters are two bytes long. So, you no longer have to worry about testing for byteness in a Unicode stream. However, all ASCII characters are represented with a leading 0x00 in Unicode. So you can't have loops that look for a terminating NULL in a C-string. And, all your formerly one-byte text doubles in size unless you explicitly compress it (and then you lose the byteness testing advantage).

Whether or not you think testing byteness is too complex or expensive to do, you should know that Unicode also does another controversial thing: For the so-called "Han" languages (Japanese, Chinese, Korean) that use characters that originated in China, it attempts to unify them into one codepoint for each character judged by the Unicode Consortium to be unique, even if it has variant forms in each language. The same is true for Arabic languages (Persian & Farsi). Because of this, you cannot tell what language a character is in just from its codepoint. Unicode was not designed to be a multi-lingual solution, in that representations of Chinese and Japanese in the same document will have overlapping character codes, requiring the OS to provide a parallel linguistically-coded data structure to render the glyph forms appropriately to each language. This might be another version of today's font/script/language relationship on Mac OS. As you can imagine, the Chinese, Japanese and Korean governments have each published competing encoding standards to Unicode, labeling the latter as something designed by foreigners who didn't understand the issues (both political and linguistic) involved in trying to make a worldwide encoding system.

Another issue about Unicode is that although it can represent 65,536 characters, there is not enough space for all the Han characters and their variants, plus all the other languages that Unicode currently supports. New languages are becoming computerized as more countries join the Digital Revolution and the Unicode Consortium cannot give space to all of them. Preferring the flat encoding model, they came up with another standard that uses four bytes per character (the ISO 10646 encoding standard). Given that on the Internet, where many languages need simultaneous support on computers, bandwidth is at a premium, I would prefer using the ISO 2022 standard of mixed-byte (7-bit and 14-bit characters) plus the escape codes that tell you what language the current stream is in to sending 32-bit characters through the wire. Since most web pages use this encoding, expect your OS to provide utilities for encoding conversion (like the Mac OS Encoding Converter debuting soon on a Mac near you).

Cross-Platform Development Issues

Going cross-platform is already complicated without having to think about internationalization. Should you have separate codebases for maximum use of each platform's

unique features? Or should you have a single codebase and use an emulation layer for the other OS's APIs? Each has its advantages, but for this article I can speak to those of you who have a joint codebase, and tell you about some of the things that the Windows platform lacks that you have to write yourself for multi-byte support and internationalization.

Windows has no Script Manager. There is no Gestalt Manager. It cannot support multiple two-byte codepages at the same time. It uses totally separate fonts for vertical and horizontal text. It supports proportional kana in Japanese, so you can't assume all two-byte characters are the same width.

If your code uses the Script Manager routines heavily, then you will have to write them yourself on the Windows side. All the convenience of the Mac OS's international routines comes very clear when you try the same things on a PC!

Also, Japan once again has its own special challenges. Until Windows came out in Japan, each computer manufacturer made its own proprietary OS and hardware. Even floppy disks were incompatible with each other. Now, most companies have adopted the Intel PC standard, but NEC continues to manufacture its own line of incompatible PCs. NEC has such a huge share of the market in Japan that it has teamed up with Microsoft to produce its own version of Windows95 for NEC. So when you buy Windows95 in Japan, you find there are three versions: MS Windows95 for Intel, MS Windows95 for NEC, and NEC Windows95 for NEC. All three versions are basically the same feature-for-feature, but the drivers are different and you need to test your application on each platform to verify compatibility.

On the hardware side, you will find that Japanese hardware is different: They use different displays, different keyboards, different printers, and different floppy formats. The drive lettering on NEC machines is different from Intel PCs: The hard disk drive is labeled 'A:' on one and 'C:' on the other. Make sure your installer isn't hard-coded to install on drive C:.

Conclusion

As we have seen, internationalization of your software on Mac OS is not very difficult to do, and it is to your benefit to try and enable as many users as possible to enter text in their own language when using your program. We have also examined Japanese localization in more depth, and demonstrated that Japanese language applications usually require some amount of new features designed specifically for that language's needs and conventions. As more markets around the world reach maturity, you can be sure that there will be ample opportunity to differentiate your product by adding locale-specific features. It is these locale-specific features that will tell your users that they are valued customers, and that their needs are being addressed in a very specific way. For your product, especially if you are in the initial designing phases, I would recommend you try to make it as easily expandable as possible. Design generic internationization into the core modules, while leaving open the opportunity to add locale-specific features for certain markets like Japan, as you see your product's market expand and rise in success.

Bibliography and Related Reading

- Apple Computer, Inc. *Inside Macintosh: Text*, Menlo Park, CA: Addison Wesley, March 1993.
- Apple Computer, Inc. "Technote OV 20, Internationalization Checklist," Cupertino, CA: Apple Computer, Inc, November 1993.
- Griffith, Tague. "Gearing Up for Asia With the Text Services Manager and TSMTE," Develop Issue 29. Cupertino, CA: Apple Computer, Inc, March 1997.
- Apple Computer, Inc. "Technote TE 531, Text Services Manager Q&As," Cupertino, CA: Apple Computer, Inc, May 1993.
- Lunde, Ken. *Understanding Japanese Information Processing*, Sebastopol, CA: O'Reilly & Associates, September, 1993.

See also Ken Lunde's home page at <http://jasper.ora.com/lunde/> for more information about multi-byte text processing on computers. MT

by Duane Murphy, Bear River Associates, Inc.

The Mac OS, STL, & Iterators

Take advantage of the C++ Standard Library Algorithms with the Mac Toolbox

Iterating The Toolbox

There are many parts of the Mac OS Toolbox that require you to search for what you need. Search for files with a certain type code. Search for a process with a particular creator code. Search for mounted volumes that are removable. The list of managers in the Mac OS that require searching goes on and on. So, we write search and loop routines to do the work.

Now wait a minute. Aren't there generic routines in the C++ Standard Template Library (STL) that can search and locate items with specific criteria? Those won't work; they only work on iterators and containers in STL, right?

In fact, the STL is very flexible and extensible. In this article, you will learn how to take advantage of STL algorithms to search Mac OS Toolbox managers. Specifically, you will learn how to write a forward iterator that satisfies the criteria for STL algorithms in order to take advantage of them.

STL Introduction

First, a word from our friendly C++ standards committee. There really isn't a Standard Template Library any more. All of the features and functions of the STL were rolled into the C++ Standard Library proper. However, history once called it the STL, and that name lives on. What we refer to in this article as the Standard Template Library is the generic algorithms, containers, and iterators that make up a large portion of the C++ Standard Library.

There are many articles and excellent books written about the Standard Template Library. We won't go into a great deal of detail here, but we will give enough background about generic algorithms, iterators, and function objects so that everyone should understand what follows. If you're experienced with using the STL for its valuable containers, like vectors, lists, and maps, you might want to skip to the next section.

STL is made up of 5 classifications of objects:

- Generic Algorithms
- Containers
- Iterators
- Function Objects
- Adapters

Generic algorithms operate on containers by using iterators. The generic algorithms can be specialized and extended by using function objects. Adapters can be used with iterators and containers to modify their behavior. We are interested in the algorithms and iterators. Because we need to specialize the algorithms, we will also use function objects.

Containers

Containers store objects. Containers own the objects that are stored in them. Containers also provide the iterators designed to iterate through the objects in the container. Container classes in the STL include vector, list, deque, set, and map. Each container has its particular feature set. These containers are invaluable for data storage in your application, but we won't be talking about them in this article.

Duane Murphy <dmurphy@bearriver.com> is a Senior Consultant for Bear River Associates, Inc. He has been programming for over 10 years; the last 7 years on the Macintosh. Duane has a special appreciation for C++. When he is not sitting in front of his computer, he can be found playing with his two daughters, unless they're sitting in front of the computer.

Adapters

Adapters are used with iterators and containers. An iterator adapter can make a forward iterator into a reverse iterator, for example. A container adapter can turn a deque into a stack. Adapters are also very useful for your application data storage requirements, but again, we won't be needing them here.

Algorithms

These are the functions that we want to take advantage of. The most often used algorithms are for_each, find, and find_if. Sometimes you might use count or count_if. for_each will execute some function (using a function object) for each element specified by a pair of iterators.

Iterators

Iterators are the objects that connect algorithms to containers. Algorithms are always written in terms of iterators. An iterator is an abstraction of a C++ pointer. In fact, any algorithm can work with a C++ pointer as the iterator.

Function Objects

Finally, a function object, also known as a functor, gives the STL an object that can modify the behavior of an algorithm. For example, the for_each algorithm will apply a function object to an iterator. The find_if and count_if algorithms use a function object as a test, known as a predicate, for searching and counting using an iterator.

Mac OS Containers

As explained above, iterators are the objects that allow algorithms to operate on the contents of a container. Algorithms don't have any direct knowledge of a container. The container is completely isolated inside the iterator. In the STL, iterators are usually provided by the container. Of course, the Mac OS doesn't know anything about the STL; so we have to build the iterators that would be provided by the Mac OS, as if it knew about the STL.

To build the iterators we must think in terms of the STL. In particular, iterators provide access to the elements of a container. We have to find the containers in the Mac OS and then figure out how to represent an iterator over that container. You can think of a container in the OS as anything that has a list or group of things that are the same type. In this article, we will use as examples: processes in the Process Manager, volumes in the file system, and files in a directory. We can think of the Process Manager as a container that has a list of processes running in the system. We can think of the file system as a container that has a list of volumes. We can think of a directory on a volume as a container of files.

There are many other examples of lists of objects in the Mac OS. Each of these can be thought of as a container. If you can access the objects in some order, then you can build an iterator to use with the algorithms in the STL. Thinking in the same terms as the STL will help you to better understand iterators.

STL Iterators

Because iterators are the interface between containers and STL algorithms, the focus of this article is on STL iterators. In particular, we would like to take advantage of the algorithms in the STL to iterate over lists or groups of objects in the Mac OS. To do this, we must know the requirements of the iterators of the Standard Template Library.

Iterators in the STL come in five categories:

- Input Iterator
- Output Iterator
- Forward Iterator
- Bi-directional Iterator
- Random Access Iterator

Input and output iterators are used to iterate over streams. Forward iterators can only increment. Bi-directional iterators can increment and decrement, while random access iterators use the index operator to dereference any location. While there is no real inheritance in the iterator categories, you can see that each iterator category has increasing functionality.

Forward Iterators

The parts of the Mac OS Toolbox that we are interested in are usually candidates for forward iterators. The STL algorithms have a few requirements for the forward iterator.

1. **Default Constructor.** A default constructor can usually be used to mark the end point of the iteration. We will see examples of this in our iterators. Additional constructors can also be defined.
2. **Copy Constructor.** Iterators are meant to be lightweight. They are, after all, an abstraction of a pointer. Iterators are almost always passed by value; therefore, a copy constructor is needed.
3. **Comparison Operator.** STL algorithms operate over a range of a container. A begin and end iterator are provided to the algorithm. The comparison operator of the iterator is used to determine when the range has been completed.
4. **Assignment Operator.** For the same reason we need a copy constructor, we need an assignment operator. Iterators are lightweight, passed by value, and are easily assigned to one another.
5. **Dereference Operator.** This is how the algorithm gets access to the container. The algorithm will use the dereference operator of the iterator to get the value of the container identified by the iterator.
6. **Pre-increment Operator.** This is the fundamental iteration function. The algorithm will iterate over the container by calling the pre-increment and post-increment operators.
7. **Post-increment Operator.** Algorithms use both the pre-increment and post-increment operators. The iterator must provide both.

Iterator Stationery

The iterator stationery provided with this articles project files (on the MacTech ftp site) gives you a head start in creating your own iterators. Most of the functionality of the iterator is provided.

You must provide some basic nuts and bolts to customize it for your use, but most of the boiler plate code is here.

The stationery is divided into three sections: the class definition, the inline functions, and the non-inline functions. The class definition will sometimes be changed; you may want to include additional constructors or additional routines to access information about the iterator. Here is the class definition from the stationery:

```
class ForwardIterator
  : public forward_iterator <Type, ptrdiff_t>
{
  public:
      // Default constructor.
      // Required by STL.
      // Can be used as a mark for the ending point of iteration.
    ForwardIterator();

    // Insert other constructors here

      // Copy constructor
      // Required by STL.
    ForwardIterator(
      const ForwardIterator & inOriginal );

      // Comparision operator.
      // Required by STL.
    bool
    operator==(
      const ForwardIterator & inRHS) const;

      // A negative comparision operator is also
      // required by STL. But utility.h includes a template
      // function that implements != in terms of ==.

      // Assignment operator
      // Required by STL.
    ForwardIterator &
    operator=(
      const ForwardIterator & inRHS);

      // Dereference operator
      // Required by STL.
    const Type &
    operator*() const;

      // Indirection operator (Optional)
      // If type is a struct or an object then include this
    const Type *
    operator->() const;

      // preincrement operator
      // Required by STL.
    ForwardIterator &
    operator++();

      // postincrement operator
      // Required by STL. Always in terms of preincrement.
    ForwardIterator
    operator++(int);  // The presence of the (int) indicates
                      // postincrement instead of preincrement.

  protected:
    Type fType;
};
```

Many of the functions provided by the stationery do not have to be changed, while others must have the code filled in. These functions provided by the stationery usually do not need to be changed:

```
inline const Type &
ForwardIterator::operator*() const
{
  return fType;
}
```

```
inline const Type *
ForwardIterator::operator->() const
{
  return &fType;
}

inline ForwardIterator
ForwardIterator::operator++(int)
{
  //The post-increment operator is always implemented
  // in terms of the pre-increment operator.
  ForwardIterator temp = *this;
  ++(*this);
  return temp;
}
```

The other functions are discussed in rest of the article. These functions often need additional code supplied to complete them.

To use the stationery you must do a search and replace on three strings:

1. Replace "ForwardIterator" with the name of your iterator.
2. Replace "fType" with the name of the basic storage element of the container you are iterating. This is the name of a data member of the iterator.
3. Replace "Type" with the type of the basic storage element of the container that you are iterating. The iterator will represent a pointer to this type.

Preparing to Write an Iterator

Before writing an iterator, there are a few things about the iterator that we'll need to know:

1. How does it iterate?

That is, what container or list is being iterated over and how do we get the next element from the container? This will help us to determine the implementation of the pre-increment operator.

2. What is the type of the iterator?

Identify the type that will be returned by the dereference operator. This is the fundamental type of the iterator; the iterator represents a pointer to this type.

3. How do we compare iterators?

Does the system or container supply some way of identifying equality? Equality in the sense of an iterator can get a little muddy. It is supposed to signify that the two iterators refer to the same element in the same container.

4. How do we recognize the end of the iteration?

This is a special case of comparing iterators. Algorithms operate on two iterators; one representing the beginning and one representing the end of the iteration. Algorithms assume that iterators can go one past the end. The end is identified by another iterator. Our iterator must have a way not only to compare to other iterators, but also to recognize that the end of the container has been reached, and iteration has occurred.

Let's look at an example to see what all this means.

A Process Iterator

Let's create an iterator that will iterate over the currently running processes. We can think of this as being an iterator for the list of processes maintained by the Process Manager. The Process Manager gives us an easy way to do this using GetNextProcess. In this example, we will create an iterator using GetNextProcess and we will use a function object to locate a specific process with a signature.

Let's answer our iterator questions.

1. How does it iterate?

The Process Manager provides GetNextProcess, which we can use to iterate over the list of processes.

2. What is the type of the iterator?

The Process Manager represents processes using a ProcessSerialNumber. The iterator will represent a pointer to a ProcessSerialNumber.

3. How do we compare iterators?

There is only one container concerned with processes, and that is the list of processes maintained by the Process Manager. Therefore, if the ProcessSerialNumbers are the same, the iterators are the same. ProcessSerialNumbers are compared using the SameProcess function provided by the Process Manager.

4. How do we recognize the end of the iteration?

GetNextProcess will return kNoProcess as the ProcessSerialNumber when the iteration is complete. GetNextProcess also wants the iteration to begin at kNoProcess. Therefore, this is a circular iterator; the beginning and the end are the same.

We start by opening the ForwardIterator.h stationery. Replace "ForwardIterator" with "BR_LProcessIterator," the name of our class. Replace "fType" with "fPSN." fPSN will hold the process serial number of the process we are currently referring to. Finally, replace "Type" with "ProcessSerialNumber," the type of fPSN as well as the type returned by a few of the functions.

Before we review the functions and fill in the code, there are a few constants that are useful:

```
const ProcessSerialNumber kBR_NoProcess
        = { 0, kNoProcess };
const ProcessSerialNumber kBR_SystemProcess
        = { 0, kSystemProcess };
const ProcessSerialNumber kBR_CurrentProcess
        = { 0, kCurrentProcess };
```

This iterator is pretty straightforward and is almost a direct interface to the Toolbox. Therefore, we are going to implement this using all inline functions. To complete the iterator we will replace the strings in the ForwardIterator.cp stationery file and copy those functions into BR_LProcessIterator.h.

The first function to look at is the constructor. Instead of a strict default constructor, we will provide a constructor with a default parameter. This way, we can iterate over all of the processes or begin iterating at some known process.

```
inline
BR_LProcessIterator::BR_LProcessIterator(
  const ProcessSerialNumber &  inPSN )
  : fPSN( inPSN )
{
}
```

All we need to do is initialize fPSN.

Next is the comparison operator. We will build the comparison operator in two steps. First, we can map SameProcess into a global comparison operator for ProcessSerialNumbers.

```
inline bool
operator==(
  const ProcessSerialNumber &  inLHS,
  const ProcessSerialNumber &  inRHS)
{
  Boolean  sameProcess = false;

  ::SameProcess( &inLHS, &inRHS, &sameProcess );

  return sameProcess;
}
```

This lets us write the iterator comparison operator

```
inline bool
BR_LProcessIterator::operator==(
  const BR_LProcessIterator &  inRHS) const
{
  return ( *(*this) == *inRHS );
}
```

We have to dereference this twice because this is a pointer to an iterator and the iterator is a pointer to a ProcessSerialNumber which means that this is a pointer to a pointer to a ProcessSerialNumber. Dereferencing this twice gives us a ProcessSerialNumber which can then be compared using the global comparison operator.

The dereference operator, indirection operator, and post-increment operator are all provided by the stationery.

The last three functions we need to complete the process iterator come from the ForwardIterator.cp stationery. Open the stationery and make the same replacements as we did in the ForwardIterator.h file: replace "ForwardIterator" with "BR_LProcessIterator," "fType" with "fPSN," and "Type" with "ProcessSerialNumber."

The stationery helps us to fill in the interface to the iterator functions, but we need to fill in the implementation.

The first function is the copy constructor. Just like the default constructor, we initialize fPSN.

```
inline
BR_LProcessIterator::BR_LProcessIterator(
  const BR_LProcessIterator &  inOriginal )
  : fPSN ( inOriginal.fPSN )
{
}
```

The assignment operator is provided by the stationery.

The pre-increment operator gets us to the next ProcessSerialNumber. We use the Process Manager function GetNextProcess.

```
inline
BR_LProcessIterator &
BR_LProcessIterator::operator++()
{
  ::GetNextProcess( &fPSN );
  return *this;
}
```

These functions are all reasonably small, so we can copy them from the .cp file into the .h file. We only need to place the inline keyword in front of each function definition.

Use This Iterator In a Sentence

How can we use this iterator to locate a specific process? Suppose we want to locate the process serial number of the Finder to send it an AppleEvent. The algorithm we want to use is find_if. The find_if algorithm takes two iterators, and a function object and returns an iterator. This is how we would like to use it:

```
const BR_LProcessIterator  theEndProcess;
BR_LProcessIterator        theProcess;

theProcess = find_if(
               ++theProcess,
               theEndProcess,
               MyFunctor() );  // What is MyFunctor()???
if ( theEndProcess == theProcess )
{
  // The Finder is not running
}
else
{
  // The Finder is running
  // theProcess is the Finder Process
}
```

Before we figure out what MyFunctor() is, we should comment on the pre-increment of theProcess in the call to find_if. Recall that this is a circular iterator. The iterator begins by pointing to no process (kBR_NoProcess). If this iterator is incremented, then we will start at the first process. find_if expects that the iterator is pointing to the beginning of the sequence. Therefore, we need to move the pointer to the first process by incrementing it first.

In STL terms, MyFunctor() is a unary predicate. A predicate returns bool true or false. The unary part means that the predicate will receive a single argument. The argument to a function object is always the dereferenced value of the iterator. Function objects do not operate on iterators directly. Function objects operate on whatever the iterator points to.

We could write a function that specifically compares the signature for the ProcessSerialNumber to the signature for the Finder or we could write a more general comparison function and use the function object adapter, bind2nd to locate any specific signature.

The function object adapter bind2nd is used to convert a binary operation into a unary operation. The first argument is passed through from the algorithm being used. The second value that is passed to the binary operator is given to the bind2nd object when it is constructed. Here is the function object that is used with bind2nd:

```
struct ProcessHasSignature
  : binary_function< ProcessSerialNumber, OSType, bool >
{
  bool
  operator()(
    const ProcessSerialNumber &  inPSN,
    const OSType &               inSignature ) const
  {
    ProcessInfoRec info;

      // initialize the fields of the process info
    info.processInfoLength= sizeof( info );
    info.processName      = 0; // not needed
    info.processAppSpec   = 0; // not needed

    OSErr err = ::GetProcessInformation( &inPSN, &info );
    bool processHasSignature = false;
    if ( noErr == err )
    {
      processHasSignature =
        ( inSignature == info.processSignature );
    }
    return processHasSignature;
  }
};
```

Notice that the function object is a struct. All that means is that everything is public. Next, notice that the only item in the struct is a function that implements the function call operator, operator(). The function will get the process information for the ProcessSerialNumber and compare the processSignature field to the process signature inSignature.

Here is how this function object is used with bind2nd

```
const OSType              kFinderType = 'MACS';
const BR_LProcessIterator theEndProcess;
BR_LProcessIterator       theProcess;

theProcess = find_if(
               ++theProcess,
               theEndProcess,
               bind2nd(
                 ProcessHasSignature(), kFinderType ) );
if ( theEndProcess == theProcess )
{
  // The Finder is not running
}
else
{
  // The Finder is running
  // theProcess is the Finder Process
}
```

bind2nd will pass the process serial number given to it by the find_if algorithm and the constant kFinderType to the operator() function of the ProcessHasSignature object. The ProcessHasSignature() statement may throw you at first. This is not a function call. This is a constructor that takes no parameters.

Listing 1 shows the example program that first uses a process iterator to print a list of the current running processes. Then the Finder process is searched for using find_if and then SimpleText is searched for using find_if.

Listing 1: ProcessIterator.cp

```
#include <string>
#include <iostream>

#include <TextUtils.h>

#include "BR_LProcessIterator.h"

void
main()
{
  cout << "Currently running files:" << endl;

  {
    BR_LProcessIterator iterator;
    BR_LProcessIterator end;
    while ( ++iterator != end )
    {
      Str32 processName;
      FSSpec processSpec;
      ProcessInfoRec info;

          // initialize the fields of the process info
      info.processInfoLength= sizeof( info );
      info.processName      = processName;
      info.processAppSpec   = &processSpec;

      OSErr err = ::GetProcessInformation( iterator, &info );
      if ( noErr == err )
      {
        char *cstrProcessName = p2cstr( processName );
        cout << cstrProcessName << endl;
      }
      else
      {
        cout << "An error occured/n";
      }
    }
  }

  cout << endl << "Locate the Finder..."  << endl;

  {
    const OSType kFinderType = 'MACS';
    const BR_LProcessIterator theEndProcess;
    BR_LProcessIterator theProcess;

    theProcess
      = find_if( ++theProcess, theEndProcess,
          bind2nd( ProcessHasSignature(), kFinderType ) );
    if ( theEndProcess == theProcess )
    {
      cout << "The Finder is NOT running." << endl;
    }
    else
    {
      cout << "The Finder is running." << endl;
    }
  }

  cout << endl << "Locate SimpleText..."  << endl;

  {
    const BR_LProcessIterator theEndProcess;
    BR_LProcessIterator theProcess;

    theProcess
      = find_if( ++theProcess, theEndProcess,
          bind2nd( ProcessHasSignature(), sigSimpleText ) );
    if ( theEndProcess == theProcess )
    {
      cout << "SimpleText is NOT running." << endl;
    }
    else
    {
      cout << "SimpleText is running." << endl;
    }
  }
}
```

A Volume Iterator

Another list that can be iterated in the Mac OS is the list of volumes maintained by the file system. You can think of the list of volumes as a container that can be iterated. You can iterate through the list of volumes, for example, to find removable volumes or simply to provide a list of available volumes.

The first step is to answer our questions about how to implement the iterator.

1. How are we iterating?

Volumes can be iterated by using PBHGetVInfo. The iteration is achieved by making successive calls to PBHGetVInfo while incrementing the ioVolIndex field.

2. What is the iterator type?

We will represent a volume with an HParamBlockRec. Actually, we could use the volumeParam variation of the HParamBlockRec, but HParamBlockRec is close enough. The iterator will be a pointer to an HParamBlockRec.

3. How do we compare iterators?

Every mounted volume has a different volume reference number. Therefore, iterators can be compared by comparing the volume reference number in the ioVRefNum field. Just like the process iterator, there is only one container; the list of volumes stored in the file system.

4. How do you compare to the end?

This is where we have to get creative. We can initialize the iterator by setting ioVRefNum to 0. A 0 value for ioVRefNum is really a logical value representing the system volume, but should not occur during our iteration. If we use the default constructor as the end marker, then we need to similarly represent the end of the iteration. Therefore, in the pre-increment operator, we will set the ioVRefNum field to 0 when there is an error. The error indication will also prevent us from incrementing an iterator that has already been completed. Of course, the code is easier than the explanation.

We begin by creating the volume iterator in the same way we created the process iterator. Open the ForwardIterator.h stationery file. Replace "ForwardIterator" with "BR_LVolumeIterator." This will be the name of our iterator class. Next, replace "fType" with "fPB." This is the field that holds the current iterated value. We don't have a real container so we provide storage for the current item. Finally, replace "Type" with "HParamBlockRec."

Before we get started adding some real code, we are going to add a couple of helpful items to this iterator. First, we are iterating over volumes. Two useful things we need to know about volumes are the volume reference number and the volume name. So we will add two accessors to this iterator.

```
    //The volume reference number
SInt16
VolumeRefNum() const;

    //The volume name (this is an internal pointer reference
    // it will change on the next iteration).
StringPtr
VolumeName() const;
```

And, due to the way PBHGetVInfo works, we need to provide a string to store the volume name. So we add

```
Str32 fVolumeName;
```

to the protected section of the class.

Now we can begin filling in the blanks. The default constructor initializes the volume name and the parameter block with appropriate values. This iterator is also circular like the process iterator. We indicate the beginning and end of the iteration with the same value. This is indicated in the constructor by setting the ioVRefNum field to 0.

```
inline
BR_LVolumeIterator::BR_LVolumeIterator()
{
  fVolumeName[0] = 0;

  fPB.volumeParam.ioCompletion   = 0;
  fPB.volumeParam.ioResult     = noErr;
  fPB.volumeParam.ioNamePtr      = fVolumeName;
  fPB.volumeParam.ioVRefNum      = 0;
  fPB.volumeParam.ioVolIndex   = 0;
}
```

The comparison operator cannot compare parameter blocks, so we will compare ioVRefNum fields. The code for the VolumeRefNum and VolumeName accessors are also inlined in the header file.

```
inline bool
BR_LVolumeIterator::operator==(
  const BR_LVolumeIterator & inRHS) const
{
  return (
    inRHS.fPB.volumeParam.ioVRefNum
      == fPB.volumeParam.ioVRefNum );
}

inline SInt16
BR_LVolumeIterator::VolumeRefNum() const
{
  return fPB.volumeParam.ioVRefNum;
}

inline ConstStr255Param
BR_LVolumeIterator::VolumeName() const
{
  return fVolumeName;
}
```

The BR_LVolumeIterator.cp file rounds out the implementation by including the copy constructor, assignment operator, and the pre-increment operator. The copy constructor copies the parameter block and the volume name fields. Also, the volume name pointer in the parameter block must be reset to the correct volume name.

```
BR_LVolumeIterator::BR_LVolumeIterator(
  const BR_LVolumeIterator & inOriginal )
  : fPB( inOriginal.fPB )
{
  PLstrcpy( fVolumeName, inOriginal.fVolumeName );
  fPB.volumeParam.ioNamePtr = fVolumeName;
}
```

The assignment operator looks pretty much the same as the copy constructor. First, an in place copy is checked before copying the fields.

```
BR_LVolumeIterator &
BR_LVolumeIterator::operator=(
  const BR_LVolumeIterator & inRHS )
{
  if ( this != &inRHS )
  {
    fPB = inRHS.fPB;
    fPB.volumeParam.ioNamePtr = fVolumeName;
    PLstrcpy( fVolumeName, inRHS.fVolumeName );
  }
  return *this;
}
```

The pre-increment operator is the key to the iterator. First, check if there was an error. We don't want to iterate past the end and an error indicates the end. Increment the ioVolIndex field and call PBGetVInfoSync. If there was an error, reset the ioVRefNum field to 0 so that this iterator will compare equal to a default iterator.

```
BR_LVolumeIterator &
BR_LVolumeIterator::operator++()
{
  if ( noErr == fPB.volumeParam.ioResult )
  {
    ++fPB.volumeParam.ioVolIndex;
    PBHGetVInfoSync( &fPB );
    if ( noErr != fPB.volumeParam.ioResult )
    {
      fPB.volumeParam.ioVRefNum = 0;
    }
  }
  return *this;
}
```

That covers it. The example in Listing 2 shows a simple loop walking through the volumes and printing out their names. Similar code could search for a characteristic of a volume. After that, we use a unary function object or predicate that tests if a volume is locked and to print a list of locked volumes. This example uses a combination of a while loop and the find_if algorithm. This is a common technique for locating multiple items in an iteration that satisfy certain criteria.

Listing 2: VolumeIterator.cp

```
#include <string>
#include <iostream>
#include "BR_LVolumeIterator.h"
void
main()
{
  cout << "Mounted Volumes" << endl;

  {
    BR_LVolumeIterator         iterator;
    const BR_LVolumeIterator end;
    while ( ++iterator != end )
    {
      string str( (char *)&( iterator.VolumeName()[1] ),
            iterator.VolumeName()[0] );
      cout << str << endl;
    }
  }
    //A unary function or predicate for testing if a volume is locked.
    // if a volume is locked.
  struct VolumeIsLocked
    : unary_function< HParamBlockRec, bool >
```

```
{
  bool
  operator()(
    const HParamBlockRec & pb ) const
  {
    bool volumeIsLocked = false;
    if ( ( pb.volumeParam.ioVAtrb & 0x0080 ) != 0 )
    {
        // Hardware lock (like a CD)
      volumeIsLocked = true;
    }
    else if ( ( pb.volumeParam.ioVAtrb & 0x8000 ) != 0 )
    {
        // Software lock
      volumeIsLocked = true;
    }
    return volumeIsLocked;
  }
};
cout << endl << "The following volumes are locked" << endl;
{
  BR_LVolumeIterator        iterator;
  const BR_LVolumeIterator end;
  bool done = false;
  while ( !done )
  {
    iterator
      = find_if( ++iterator, end, VolumeIsLocked() );
    done = ( iterator == end );
    if ( !done )
    {
      string str((char *)&(iterator.VolumeName()[1]),
                 iterator.VolumeName()[0] );
      cout << str << endl;
    }
  }
}
```

A File Iterator

The last example iterator is a file iterator. This iterator will iterate over all of the files in a given folder. We must answer a few questions about the iterator before we get started.

1. How are we iterating?

   We can iterate over files in a folder by using PBGetCatInfo.

2. What is the iterator type?

   PBGetCatInfo uses a CInfoPBRec to iterate over the files. However, FSSpecs are easier to use in other Toolbox calls. A CInfoPBRec will be used internally, but an FSSpec will be used externally.

3. How do we compare iterators?

   Iterators are equal if their ioVRefNum, ioDirID, and ioFDirIndex fields are the same. That is, if they are referring to the same volume, the same directory, and the same indexed file. The volume and directory represent the container, while the file represents the element in the container.

4. How do we recognize the end?

   The ioFDirIndex field is initialized to 0 and incremented through the files. When an error such as iterating past the last file occurs, the ioFDirIndex field is reset to 0. A special case in the comparison operator is to check for the ioFDirIndex fields both being equal to 0.

We begin constructing the iterator the same way. The name of the iterator is BR_LFileIterator. The name of the iteration field is fSpec and its type is FSSpec. Three Replace-All's and we are most of the way there.

First, add a couple of constructors that are pretty useful:

```
  // Construct from an FSSpec
  // Note the use of explicit. This constructor is not a type converter.
explicit
BR_LFileIterator(
  const FSSpec & inFolderSpec);
  // Construct from vRefNum and DirID
BR_LFileIterator(
  short  inFolderVRefNum,
  long   inFolderDirID);
```

The underlying implementation uses a CInfoPBRec and we might want to use it in other places so:

```
  // Get access to the CInfoPBRec.
const CInfoPBRec &
GetCInfoPBRec( void ) const;
```

Finally, add the parameter block:

```
CInfoPBRec fPB;
```

Next, fill in the inline functions in the header file. The default constructor turns out to be a bit too much to be efficient

inline, so it should be moved into BR_LFileIterator.cp after we create it from the stationery file. Likewise, the other constructors will also be implemented in BR_LFileIterator.cp.

The only function to fill in here is the comparison operator; the other functions work just fine from the stationery file. The comparison operator looks like:

```
inline bool
BR_LFileIterator::operator==(
  const BR_LFileIterator &inRHS) const
{
  // If both indexes are 0 then these are equal.The only time
  // an index is zero is at the beginning or the end of an iteration.
  bool equal =
      ( inRHS.fPB.hFileInfo.ioFDirIndex   == 0 )
    && ( fPB.hFileInfo.ioFDirIndex        == 0 );
  if ( !equal )
  {
    // Otherwise all three of index, directory, and vRefNum
    // must be equal.
    equal =
      ( inRHS.fPB.hFileInfo.ioFDirIndex
        ==  fPB.hFileInfo.ioFDirIndex )
    && ( inRHS.fPB.hFileInfo.ioDirID
        == fPB.hFileInfo.ioDirID )
    && ( inRHS.fPB.hFileInfo.ioVRefNum
        == fPB.hFileInfo.ioVRefNum );
  }
  return ( equal );
}
```

This is a little complicated, but not overly so. First, check for the index values being 0. The only time the index values are zero is before the beginning or after the end of the iteration. We treat these points the same, so if both indexes are zero, then the iterators are equal.

Otherwise, we compare the index, directory ID, and the volume reference numbers all for equality. If they are all equal then the iterators are also equal.

Next, create BR_LFileIterator.cp. Begin by opening the ForwardIterator.cp stationery file. Make the same replacements here as we did in the header file: "ForwardIterator" is "BR_LForwardIterator," "fType" is "fSpec," and "Type" is "FSSpec."

The first three functions are constructors. They each just set default values for fields or copy the relevant fields from the source. The default constructor initializes the relevant fields to zeros.

```
BR_LFileIterator::BR_LFileIterator()
{
  fSpec.vRefNum = 0;
  fSpec.parID   = 0;
  fSpec.name[0]   = 0;

  fPB.hFileInfo.ioCompletion = 0;
  fPB.hFileInfo.ioResult     = noErr;
  fPB.hFileInfo.ioNamePtr      = fSpec.name;
  fPB.hFileInfo.ioVRefNum      = 0;
  fPB.hFileInfo.ioFDirIndex    = 0;
  fPB.hFileInfo.ioDirID      = 0;
}
```

The copy constructor copies the structures wholesale, and then fixes the ioNamePtr field in the parameter block to point to the correct FSSpec.

```
BR_LFileIterator::BR_LFileIterator(
  const BR_LFileIterator &inOriginal )
  : fSpec( inOriginal.fSpec ),
    fPB( inOriginal.fPB )
{
  // Fix the ioNamePtr field in the parameter block.
  fPB.hFileInfo.ioNamePtr = fSpec.name;
}
```

The constructor from a volume reference number and directory ID just use the values to initialize the FSSpec and parameter block fields.

```
BR_LFileIterator::BR_LFileIterator(
  short  inFolderVRefNum,
  long inFolderDirID)
{
  fSpec.vRefNum= inFolderVRefNum;
  fSpec.parID  = inFolderDirID;
  fSpec.name[0] = 0;

  fPB.hFileInfo.ioCompletion = 0;
  fPB.hFileInfo.ioResult     = noErr;
  fPB.hFileInfo.ioNamePtr      = fSpec.name;
  fPB.hFileInfo.ioVRefNum      = fSpec.vRefNum;
  fPB.hFileInfo.ioFDirIndex    = 0;
  fPB.hFileInfo.ioDirID      = fSpec.parID;
}
```

Constructing from an FSSpec is a little more complicated. If the FSSpec is for a folder, then we need its directory ID. If the FSSpec is to a file, then we needs its parent directory ID. First, initialize the parameter block and FSSpec fields. Then call PBGetCatInfoSync. Checking the ioFlAttrib field tells us whether the result is a directory or a file. We can then initialize the parID field of the FSSpec with the correct field value.

```
BR_LFileIterator::BR_LFileIterator(
  const FSSpec & inFolderSpec)
{
  // Copy the fsSpec into our FSSpec.
  fSpec.vRefNum= inFolderSpec.vRefNum;
  fSpec.parID  = inFolderSpec.parID;
  PLstrcpy( fSpec.name, inFolderSpec.name );
  // Initialize the fields of the parameter block
  fPB.hFileInfo.ioCompletion = 0;
  fPB.hFileInfo.ioNamePtr      = fSpec.name;
  fPB.hFileInfo.ioVRefNum      = fSpec.vRefNum;
  fPB.hFileInfo.ioFDirIndex    = 0;
  fPB.hFileInfo.ioDirID      = fSpec.parID;
  // Get the info for the FSSpec.
  //
  // If the FSSpec was a folder then the ioDrDirID field is
  // the directory ID of that folder and this will iterate files
  // in that directory.
  //
  // If the FSSpec was to a file, then the ioFlParID field
  // is the directory ID of the parent directory and this will
  // iterate files in that directory.
  //
  // Maintain the directory id in the parID field of the FSSpec.
  // PBGetCatInfo will over write the value, so it must be restored
  // before each call. See operator++().
  OSErr err = ::PBGetCatInfoSync( &fPB );
  bool isDirectory
        = ( fPB.hFileInfo.ioFlAttrib & ioDirMask ) != 0;
  fSpec.parID = isDirectory ?
          fPB.dirInfo.ioDrDirID : fPB.hFileInfo.ioFlParID;
}
```

The assignment operator is straightforward. Instead of copying the structures wholesale, we'll only copy the fields we need. We must remember to set the ioNamePtr field appropriately.

```
BR_LFileIterator &
BR_LFileIterator::operator=(
  const BR_LFileIterator &inRHS )
```

```
{
  if ( this != &inRHS )
  {
    // Copy the FSSpec
    fSpec.vRefNum = inRHS.fSpec.vRefNum;
    fSpec.parID  = inRHS.fSpec.parID;
    ::PLstrcpy( fSpec.name, inRHS.fSpec.name );
    // Copy the parameter block.We only care about 6 fields.
    fPB.hFileInfo.ioCompletion = 0;
    fPB.hFileInfo.ioResult
      = inRHS.fPB.hFileInfo.ioResult;
    fPB.hFileInfo.ioNamePtr
      = fSpec.name;
    fPB.hFileInfo.ioVRefNum
      = inRHS.fPB.hFileInfo.ioVRefNum;
    fPB.hFileInfo.ioFDirIndex
      = inRHS.fPB.hFileInfo.ioFDirIndex;
    fPB.hFileInfo.ioDirID
      = inRHS.fPB.hFileInfo.ioDirID;
  }
  return *this;
}
```

Finally, the real iteration part of the iterator, the pre-increment operator:

```
BR_LFileIterator &
BR_LFileIterator::operator++()
{
    // Only increment if there is no error.
  if ( noErr == fPB.hFileInfo.ioResult )
  {
    ++fPB.hFileInfo.ioFDirIndex;    // increment the index
    fPB.hFileInfo.ioDirID = fSpec.parID;  // use the correct dir ID
    OSErr err = ::PBGetCatInfoSync( &fPB );
    if ( noErr != err )
    {
        // Reset the index to indicate that we are done.
      fPB.hFileInfo.ioFDirIndex = 0;
    }
  }
  return *this;
}
```

First, make sure that there has not been an error. Then we increment the index. We also need to copy the directory ID field. This field is an IO field for PBGetCatInfoSync so we must reset it before each call. Next, call PBGetCatInfoSync. If an error occurs, we reset the index to zero to indicate that we are done.

That completes our file iterator. This iterator can iterate over any folder returning all of the files and folders in that folder. Shown in Listing 3 is an example program that uses a simple while loop to iterate over all of the files in the System Folder.

After that, we use a function object called FSSpecIsType to locate all of the 'INIT' files in the extension folder. This example also uses a combination of a while loop and the find_if algorithm as we did in the volume iterator example. This is a common technique for locating multiple items in an iteration that satisfy certain criteria. If we needed the list of 'INIT' files to be persistent, we could use remove_copy_if and a vector of FSSpecs.

Listing 3: FileIterator.cp

```
#include <Folders.h>
#include "BR_LFileIterator.h"
void
main()
{
  short  vRefNum  = 0;
  long   dirID    = 0;
  FindFolder(
    kOnSystemDisk, kSystemFolderType,  kDontCreateFolder,
    &vRefNum, &dirID );
  BR_LFileIterator  end; // just a default marker
  BR_LFileIterator  iterator( vRefNum, dirID );
  cout << "List the files in the system folder\n";
  while ( ++iterator != end )
  {
    string str(
      (char *)&(iterator->name[1]), iterator->name[0] );
    cout << str << endl;
  }
  cout << "List the INITs in the extension folder\n";
    // Define a binary function that tests
    // the file type of an FSSpec.
  struct FSSpecIsType
    : binary_function< FSSpec, OSType, bool >
  {
    bool
    operator()(
      const FSSpec & inSpec,
      const OSType & inType ) const
    {
      bool isType = false;
      FInfo fInfo;
      OSErr err = FSpGetFInfo( &inSpec, &fInfo );
      if ( noErr == err )
      {
        isType = ( inType == fInfo.fdType );
      }
      return isType;
    }
  };
  FindFolder(
    kOnSystemDisk, kExtensionFolderType, kDontCreateFolder,
    &vRefNum, &dirID );
  FSSpec extensionFolder;
  FSMakeFSSpec( vRefNum, dirID, 0, &extensionFolder );
  BR_LFileIterator extIterator( extensionFolder );
  bool done = false;
  while ( !done )
  {
    extIterator = find_if( ++extIterator, end,
      bind2nd( FSSpecIsType(), 'INIT' ) );
    done = ( extIterator == end );
    if ( !done )
    {
      string str(
        (char *)&(extIterator->name[1]),
            extIterator->name[0] );
      cout << str << endl;
    }
  }
}
```

Conclusion

We hope that you have learned how you can take advantage of the Standard Template Library to easily find information and iterate over items in the Mac OS. There are many other iterators that can be created. Some ideas are to iterate over items in an AppleEvent list, extend the file iterator to iterate over folder hierarchies, or iterate over the items in a resource list such as a string list ('STR#') resource. We are sure you will come across many other examples of iterators in your programming. The stationery files and the techniques described here should help you to construct iterators for your applications.

Want to suggest an article for the magazine? Send your suggestion to <mailto:editorial@mactech.com>

by Guillaume Bédard, Frédéric Leblanc, Yohan Plourde and Pierre Marchand, Québec, Canada

Fast Square Root Calculation

Optimizing PowerPC Assembler Code to beat the Toolbox

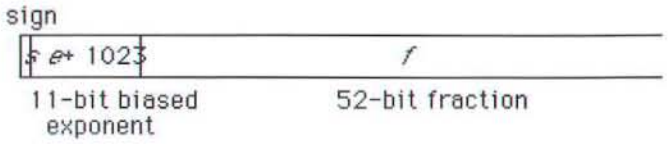
Introduction

The calculation of the square root of a floating-point number is a frequently encountered task. However, the PowerPC processors don't have a square root instruction. The implementation presented here performs the square root of a double-precision number over the full range of representation of the IEEE 754 standard for normalized numbers (from 2.22507385851E-308 to 1.79769313486E308) with an accuracy of 15 or more decimal digits. It is very fast, at least six times faster than the Toolbox ROM call.

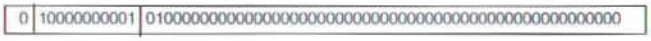
Theory of Operation

A floating point number has three components: a sign, a mantissa and an exponential part. For example, the number $+3.5 \times 10^4$ (35 000) has a plus sign, a mantissa of 3.5 and an exponential part of 10^4. The mantissa consists of an integer part and a fractional part *f*.

A double precision number in IEEE 754 format has the same components: a sign bit **s**, an 11-bit exponent ***e*** and a 52 bit fraction ***f***. The exponential part is expressed in powers of 2 and the exponent is *biased* by adding 1023 to the value of *e*. The mantissa is *normalized* to be of the form 1.*f*. Since the integer part of the normalized mantissa is always 1, it doesn't have to be included in the representation. The number is thus represented as follows: $(-1)^s \times 1.f \times 2^{e+1023}$.



For example, the number 5.0 can be expressed in binary as 101.0, which means 101.0×2^0, which in turn is equal to 1.010×2^2, obtained by dividing the mantissa by 4 and multiplying 2^0 by 4. Therefore, the normalized mantissa is 1.010 and the exponent 2. Fraction *f* is then .01000000.... The biased exponent is obtained by adding 1023 to *e* and is 1025, or 10000000001 in binary. The double precision IEEE representation of 5.0 is finally:



or 4014000000000000 in hexadecimal notation for short.

First Approximation

Given this representation, a first approximation to the square root of a number is obtained by dividing the exponent by 2. If the number is an even power of 2 such as 16 or 64, the exact root is obtained. If the number is an odd power of 2 such as 8 or 32, $1/\sqrt{2}$ times the square root is obtained. In general, the result will be within a factor $\sqrt{2}$ of the true value.

Refining the Approximation

The Newton-Raphson method is often used to obtain a more accurate value for the root *x* of a function $f(x)$ once an initial approximation x^0 is given:

[1]

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

The first three authors are undergraduate students in Computer Science at Université Laval in Québec, Canada. This work was done as an assignment in a course on Computer Architecture given by the fourth author.

This becomes, in the case of the square root of n, = x^2 - n:where $f(x)$

[2]
$$x = \frac{1}{2}\left(x_0 - \frac{n}{x_0}\right)$$

An excellent approximation to the square root starting with the initial approximation given above is obtained within 5 iterations using equation [2]. This algorithm is already pretty fast, but its speed is limited by the fact that each iteration requires a double-precision division which is the slowest PowerPC floating-point instruction with 32 cycles on the MPC601 (Motorola, 1993).

Eliminating divisions

Another approach is to use equation [1] with the function.

$$f(x) = \frac{1}{x^2} - \frac{1}{n}$$

In this case, equation [1] becomes:

[3]
$$x = \frac{3}{2} x_0 \left(1 - \frac{x_0^2}{n}\right)$$

There is still a division by n, but since n is constant (it's the original number whose root we want to find), it can be replaced by multiplying by $1/n$, which can be calculated once before the beginning of the iteration process. The five 32-cycle divisions are thus replaced by this single division followed by 5 much faster multiplications (5 cycles each). This approach is approximately three times faster than the preceding one. However, care must be taken for large numbers since the term in x^{02} can cause the operation to overflow.

Use of a table

Finally, an approach that is even faster consists in using a table to obtain a more accurate first approximation. In order to do so, the range of possible values of fraction f (0 to ~1) is divided into 16 sub-ranges by using the first 4 bits of f as an index into a table which contains the first two coefficients of the Taylor expansion of the square root of the mantissa (1.0 to ~2) over that sub-range.

The Taylor expansion is given in general by:

[4]
$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)(x - x_0)^2}{2!} + \ldots$$

the first two terms of which yield, in the case where $f(x) = \sqrt{x}$:

[5]
$$\sqrt{x} \cong \sqrt{x_0} + \frac{(x - x_0)}{2\sqrt{x_0}} = \frac{\sqrt{x_0}}{2} + \frac{1}{2\sqrt{x_0}} x$$

The square root of x is thus approximated by 16 straight-line segments. The table therefore contains the values of

$$A = \frac{\sqrt{x_0}}{2}$$

and $B = \dfrac{1}{2\sqrt{x_0}}$

for each of the 16 sub-ranges as shown in **Figure 1**. This first approximation gives an accuracy of about 1.5 %.

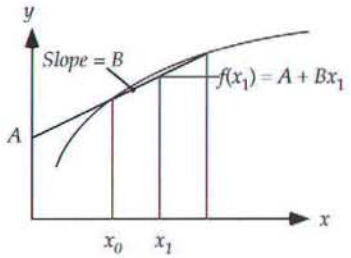


Figure 1. *Approximation by straight line segment.*

To reach the desired accuracy of 15 digits, equation [2] is applied twice to the result of equation [5]. To avoid having to perform two divisions by repeating the iteration, the two iterations are folded together as follows, which contains only one division:

and

[6]
$$x'' = \frac{1}{2}\left(x' + \frac{n}{x'}\right) = \frac{(x^2 + n)^2 + 4x^2 n}{4x(x^2 + n)}$$

In order to perform these calculation, the exponent of x and n is reduced to -1 (1022 biased), so that floating-point operations apply only to the values of the mantissa and don't overflow if the exponent is very large. The value of these numbers will therefore be in the range 0.5 to 1.0 since the mantissa is in the range 1.0 to 2.0. If the original exponent was odd, the mantissa is multiplied by $\sqrt{2}$ before applying equation [6].

Finally, the original exponent divided by two is restored at the end.

The Code

The SQRoot function shown in Listing 1 has been implemented in CodeWarrior C/C++ version 10.

Listing 1: SQRoot.c

```
// On entry, fp1 contains a positive number between 2.22507385851E-308
// and 1.79769313486E308. On exit, the result is in fp1.

asm long double SQRoot(long double num);  // prototype

float Table[35] = {
0.353553390593, 0.707106781187, 0.364434493428, 0.685994340570,
0.375000000000, 0.666666666667, 0.385275875186, 0.648885684523,
0.395284707521, 0.632455532034, 0.405046293650, 0.617213399848,
0.414578098794, 0.603022689156, 0.423895623945, 0.589767824620,
0.433012701892, 0.577350269190, 0.441941738242, 0.565685424949,
0.450693909433, 0.554700196225, 0.459279326772, 0.544331053952,
0.467707173347, 0.534522483825, 0.475985819116, 0.525225731439,
0.484122918276, 0.516397779494, 0.492125492126, 0.508000508001,
1.414213562373, 0.000000000000, 0.000000000000 };

asm long double Sqrt(long double num) {

   lwz  r3,Table(rtoc)     // address of Table[]
   lhz  r4,24(sp)          // load
                           // Sign(1)+Exponent(11)+Mantissa(4)
   andi.  r5,r4,0xF        // keep only Mantissa(4)
   ori  r5,r5,0x3FE0       // exponent = -1+BIAS = 1022
   sth  r5,24(sp)          // save reduced number

   rlwinm r5,r5,3,25,28    // take 8*Mantissa(4) as index
   lfd  fp1,24(sp)         // load reduced number
   lfsux  fp4,r5,r3        // load coefficient A
   lfs  fp5,4(r5)          // load coefficient B
   lfs  fp3,128(r3)        // load SQRT(2)
   fmr  fp2,fp1            // copy reduced number
   rlwinm.  r5,r4,31,18,28 // divide exponent by 2
   beq  @@2                // if (exponent == 0) then done

   fmadd  fp2,fp2,fp5,fp4 // approximation SQRT(x) = A + B*x
   andi.  r4,r4,0x10       // check if exponent even
   beq  @@1                // if (exponent even) do iteration
   fmul fp2,fp2,fp3        // multiply reduced number by SQRT(2)
   fadd fp1,fp1,fp1        // adjust exponent of original number

@@1: fadd fp3,fp2,fp2      // 2*x
   fmul fp5,fp2,fp1        // x*n
   fadd fp3,fp3,fp3        // 4*x
   fmadd  fp4,fp2,fp2,fp1 // x*x + n
   fmul fp5,fp3,fp5        // 4*x*x*n
   fmul fp6,fp2,fp4        // denominator = x*(x*x + n)
   fmadd  fp5,fp4,fp4,fp5 // numerator = (x*x + n)*(x*x + n) +
                           // 4*x*x*n
   fdiv fp1,fp5,fp6        // double precision division
   andi.  r5,r5,0x7FF0     // mask exponent
   addi r5,r5,0x1FE0       // rectify new exponent

@@2: sth  r5,132(r3)       // save constant C (power of 2)
   lfd  fp2,132(r3)        // load constant C
   fmul fp1,fp1,fp2        // multiply by C to replace exponent
   blr                     // done, the result is in fp1
}
```

Performance

The code presented above runs in less than 100 cycles, which means less than 1 microsecond on a 7200/75 Power Macintosh and is more than six times faster than the ROM code. The code could be modified to make use of the *floating reciprocal square root estimate* instruction (frsqrte) that is available on the MPC603 and MPC604 processors, and which has an accuracy of 5 bits. It is not available on the MPC601, however. The method used here could also be used to evaluate other transcendental functions.

Performance was measured by running the code a thousand times and calling a simple timing routine found in (Motorola, 1993), that we called myGetTime(). It uses the real-time clock of the MPC 601 processor (RTCU and RTCL registers) and is shown in Listing 2. The routine would have to be modified to run on MPC603 or MPC604 processors, since they don't have the same real-time clock mechanism.

The code doesn't support denormalized numbers (below 2.22507385851E-308). This could easily be implemented albeit at the cost of a slight reduction in performance.

Listing 2: myGetTime.c

```
asm long myGetTime()
  {
lp: mfspr  r4,4          // RTCU
  mfspr  r3,5            // RTCL
  mfspr  r5,4            // RTCU again
  cmpw   r4,r5           // if RTCU has changed, try again
  bne    lp
  rlwinm r3,r3,25,7,31 // shift right since bits 25-31 are
         // not used
  blr                    // the result is in r3. 1 unit is
         // worth 128 ns.
  }
```

To run the code, a very simple interface using the SIOUX library is provided in Listing 3.

Listing 3: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fp.h>

void main()
  {
  long double num, num2;
  long startTime, endTime, time;
  short i;


  do {
  printf("%2s","> ");              // caret
  scanf("%Lf",&num);               // read long double
  if (num < 0.0) num = 0.0;        // replace by 0.0 if negative
  startTime = myGetTime();
  for (i = 0; i < 1000; i++)       // repeat 1000 times
  num2 = SQRoot(num);              // call our function
  endTime = myGetTime();
  time = endTime - startTime;
  if (num > 1e-6 && num < 1e7)
    printf("%7s%Lf\n","root = ",num2);  // show result
  else
    printf("%7s%Le\n","root = ",num2);
  printf("%7s%d\n","time = ", time);     // show elapsed time
  }
  while (1);                       // repeat until Quit
  }
```

References

PowerPC 601 RISC Microprocessor User's Manual, Motorola MPC601UM/AD Rev 1, 1993.

by Dave Mark,

CodeWarrior Latitude: Porting Your Apps to Rhapsody

The June, 1997 Factory Floor column contained an interview with David Hempling, CodeWarrior Latitude engineering lead. You can find the interview at: <http://www.metrowerks.com/products/cw/latitude/index.html>.

For those who missed the interview, CodeWarrior Latitude is essentially a porting library that greatly simplifies the process of porting MacOS applications to Rhapsody. Latitude is an implementation of the Macintosh System 7 application programming interfaces (APIs) that allows source code for a Macintosh application to be compiled with little modification on a Rhapsody platform, resulting in a native Rhapsody application. Latitude maps the Mac API requests to native Unix system calls. An application built with Latitude attains the look and feel of the new native platform. In this column we'll explore the process of using Latitude to port applications to Rhapsody.

A Simple Example

Let's start off with a simple example, straight from the pages of the Mac Primer, Volume II. ColorTutor allows you to play with the various Color Quickdraw drawing modes. Written in C, ColorTutor consists of a project file, a C source file, and a resource file containing MENU & WIND resources.

The first task in porting an application to Rhapsody is to physically get the application's files to the Rhaposdy platform. There are several ways to do this. For the source code itself, any means of transferring text files from one computer to another will do. If we had a collection of sources, we could tar them up with a tar tool on the Mac, ftp the tar file to Rhapsody with something like Fetch, and untar the image on the other side. This method *cannot* be used to transfer Mac resource data.

Another method would be to use one of several available UNIX/Mac file sharing systems such as MacNFS, NFS/Share, IPT uShare, Helios EtherShare, or Xinet K-AShare. Latitude's File Manager understands the various formats that these systems use so they are also good for transferring Mac resource data.

Before copying over the resource file, however, we must ensure that a BNDL resource is present in the application's resource data. Latitude requires the signature value in an application's BNDL resource in order to maintain its own "desktop database". ColorTutor.rsrc does not have a BNDL resource so we'll add one, using ResEdit, on the Mac. We'll make ColorTutor's signature 'CTTR'. That done, we'll build ColorTutor on the Mac, ensuring that all of the application's resources are collected into the application's executable.

For this example, we use NFS/Share to transfer the ColorTutor executable to our target platform. NFS/Share uses AppleDouble format, so the file will be split into two pieces when it is transferred, namely ColorTutor and %ColorTutor.

Once we've successfully transferred the file to our Rhapsody file system, we place %ColorTutor in a directory named:

```
$LATITUDE_ROOT/Latitude.MacFiles/Applications/ColorTutor/
```

$LATITUDE_ROOT is where Latitude was installed on our Rhapsody system. We place ColorTutor.c in the directory:

```
$LATITUDE_ROOT/Latitude.MacFiles/Applications/
    ColorTutor/Sources/
```

We can now prepare the ColorTutor.c file for compilation. Latitude includes a tool called prepare_sources that traverses your source files and replaces non-ANSI Mac-isms in your sources (such as pascal strings, eight-bit characters, and four character constants) with macros that conditionally expand to either the Macintosh or the ANSI styles, depending on where the build takes place. It also creates a Makefile usable with gnu make that can be used to build your application.

prepare_sources wants to know the name of the application we're creating and the type of system includes it is using:

```
% prepare_sources ColorTutor universal
Creating Latitude.manifest...
Converting source files to Latitude files.
mac2unix: processing ./ColorTutor.c
Writing default Makefile...
Conversion complete.
```

ColorTutor.c is copied to ColorTutor.c.bak, preserving the original code, and three files are created: the modified source ColorTutor.c, a Makefile, and Latitude.Manifest, which is a list of the source files used to create the application.

Looking at ColorTutor.c, we can see that prepare_sources replaced all "\p..." strings with PSTRINGCONST("...") macros. It also replaced four-character constants like 'DRVR' with the QUADCONST('D','R','V','R') macro. Even though some non-Mac compilers will handle a four character constant, the order of the characters in the resulting long can be flipped. By using QUADCONST(), we ensure that the proper ordering is achieved.

When compiled on the Mac, ColorTutor.c relied on the MacHeaders pre-compiled headers. Since we don't have pre-compiled headers on our non-Mac platform (yet), we must explicitly include MacHeaders in our source. A copy of the standard MacHeaders is included in $LATITUDE_ROOT/utilities, so we'll copy it to our ColorTutor/Sources and #include it at the top of ColorTutor.c

ColorTutor.c uses qd. to access Quickdraw low memory globals like thePort, screenBits, etc. CodeWarrior Latitude supports these globals but does not keep them in a qd structure. We can use a macro to remove qd. when it is used. At the top of ColorTutor.c, we add the following:

```
#ifdef _LATITUDE_
#define QD(x) x
#else
#define QD(x) qd.x
#endif
```

We now go through ColorTutor.c and replace instances of qd. with the use of the QD() macro. So

```
InitGraf( &qd.thePort );
```

becomes

```
InitGraf(&QD(thePort));
```

There are a few other instances in the file which we'll replace as well.

Before we build ColorTutor, we must insert code to initialize Latitude. The lg_latitude_init() call does several things. It sets up Latitude's trap table, initializes various toolbox managers, and opens Latitude's System file and the application's resource file. lg_latitude_init() also allows settings that affect how the application looks and behaves. For ColorTutor, only the bare minimum is required.

Currently, main() starts out like this:

```
void    main( void )
{
        ToolboxInit();
        MenuBarInit();
```

We're going to change main() to accept arguments and pass those arguments to lg_latitude_init:

```
#ifndef _LATITUDE_
void    main( void )
{
#else
void    main( long argc, char **argv )
{
#ifdef _LATITUDE_
        LG_APP_INFO_BLOCK lblock;

        lblock.flags = LG_APP_INFO_SIGNATURE |
           LG_APP_INFO_CLASSNAME;
        lblock.signature = QUADCONST('C','T','T','R');
        lblock.classname = "Latitude";
        lg_latitude_init(argc,argv, &lblock);
#endif

        ToolboxInit();
        MenuBarInit();
```

We're now ready to build ColorTutor. Using the gcc 2.7.2 compiler, this code compiles without error. However, if you're using a different C compiler, you may find that the C++ style '//' comments at the top of ColorTutor.c are a problem. If so, simply replace them with C style '/* */' comments and rebuild.

The first time we run ColorTutor, Latitude brings up a Get File Dialog asking us to locate the application's resource file. This is only done once. Latitude stores the location of the resource file in it's own desktop database located at:

```
user.MacFiles/System/Preferences/LatitudeApps
```

Once that is done, ColorTutor is off and running on our platform. Playing with the app's popup menus and seeing the proper results in the dialog shows that this application was ported successfully with very little effort. Additionally, we can take this source code back to the Mac, along with the LGLatitude.h file, and continue to use the same source to build both our Mac and non-Mac platform versions.

PowerPlant

Obviously, ColorTutor is a very simple application that doesn't contain any serious portability problems. It's ANSI C code, contains no private resource data that it uses itself, doesn't patch traps, doesn't have custom definition functions, etc. Lets move on to a more complex porting example and look at what's required to bring a PowerPlant app, like Muscle, over to Rhapsody using CodeWarrior Latitude.

The first thing to notice about PowerPlant is that it's written in C++ and that its source code is laid out hierarchically in a directory structure. It's easy to replicate this structure on our new platform. The prepare_sources script currently only recognizes file extensions for C (.c & .h). It is easy to modify the 'find' incantation in prepare_sources to recognize the C++ file naming convention used in PowerPlant. prepare_sources is also fairly naive about complex development environments. Depending on your own expertise with Makefiles, you may decide to build the development environment yourself. You do need to run the conversion script, mac2unix, over the code to take care of the pascal strings and four character constants, however.

Inside the PowerPlant code itself, there are a few modifications we need to make.

Template Instantiation

Template instantiation is not standardized across C++ platforms. When we brought PowerPlant away from the Metrowerks environment, we entered a world where multiple instantiations of C++ templates needed to be handled differently.

There are two classic approaches to solving the instantiation issue: the Cfront and Borland models. In Cfront all instantiations occur in a single well-known directory. The Borland model, on the other hand, instantiates each template in each translation unit with duplicates being removed by the linker. Both techniques have serious flaws and have led to other approaches. The gcc compiler that we're using on Rhapsody, in particular, eschews these and provides instead 3 different techniques. The details of the problems encountered with all of these approaches will be deferred for a future article. In short, the only approach that worked was to:

- disable implicit instantiation (compiler option)
- disable explicit instantiations in header files
- explicitly instantiate each template in each translation unit in which the linker complained

This solution is not optimal because the burden for template instantiation is placed on the programmer. Change the code or how the translation units are combined and you may have to change where the explicit instantiations are placed. But it does work. We hope that better automatic approaches are found in the future that eliminates the need for any explicit instantiation statements.

For example, we had to turn off any explicit instantiation statements that occurred in .h files (really a bad idea to begin with):

```
Pane_Classes/LView.h

    #if !defined(MW__GCC_EXPLICIT_TEMPLATE_INSTANTIATION)
    template class TArray<LPane*>;
    #endif
```

and then to make sure that source files that required the instantiations had them:

```
Pane_Classes/LView.cpp

    #if defined(MW__GCC_EXPLICIT_TEMPLATE_INSTANTIATION)
    template class TArray<LPane*>;
    template class TArrayIterator<LPane*>;
    #endif
```

Scripting & AEObject Support

Latitude's Apple event implementation is not complete. While basic Apple event calls are in place, AEObject support is not. PowerPlant uses AEObjects to record user actions as well as trigger window management, such as taking a window down or allowing a window to be dragged. PowerPlant's AEObject code is centralized and it was clear where an event was dispatched and where it was picked up. We modified the code to perform the end result directly, without going through the AEObject dispatching mechanism.

For example, PowerPlant's LWindow::AttemptClose() function now looks like this:

```
void
LWindow::AttemptClose()
{
  // Get approval from SuperCommander
  if ((mSuperCommander == nil)
    || mSuperCommander->AllowSubRemoval(this)) {

    // Send Close AE for recording only
#ifdef MW__LATITUDE_AE_OBJECT_SUPPORT
    DoClose();
#else
    SendSelfAE(kAECoreSuite, kAEClose, false);
    delete this;
#endif
  }
}
```

Long-Word Alignment

Mac resource data is kept in 680x0 alignment space, meaning long words can fall on short word boundaries. Reading in a 680x0 aligned resource on a platform in which long words are aligned on long word boundaries yields inaccessible data and even segmentation and bus errors, depending upon how the source code accesses the data. While some compilers have align pragmas that can keep structures in a specified alignment space, these pragmas are not common to all platforms, even with the same compiler. We recommend that these pragmas not be used since they don't ensure portability.

Latitude includes tools to pack and unpack Mac resources and file data. By using these tools, your app is assured to be portable to other platforms where CodeWarrior Latitude is supported. Additionally, Latitude's lg_align* tools will perform byte swapping once Latitude is made available for x86 platforms.

In PowerPlant, the LWindow::MakeMacWindow() function sneaks a peek at a WIND resource before calling Mac Window Manager's GetNewWindow(). The code fragment looks like this:

```
SWINDREsourceH theWind = (SWINDResourceH)
::GetResource(QUADCONST('W','I','N','D'), inWINDid);

Int16 kind;

#ifdef _LATITUDE_
if (lg_align_unpack((Handle)theWIND, SWINDResource_align) !=
noErr)
{
  /* unpack failed! */
}
#endif
kind = (**theWIND).refcon;
```

The SWINDResource_align argument to lg_align_unpack is a LG_ALIGN token array containing a description of the elements of the WIND resource. Had we not done the lg_align_unpack, the kind variable would contain garbage since the refcon field of a WIND resource is not naturally long word aligned.

Floating Windows

Many Mac applications have floating windows and it seems that every Mac application accomplishes this feat differently. Some patch traps such as FrontWindow, SendBehind, and WaitNextEvent. Some modify the Window Manager's WindowList. Latitude is capable of supporting floating windows through many different methods. The best method, however, is to let Latitude float the windows without intervention.

Regardless of how your application floats windows, it is best to alert Latitude that a certain window is meant to float when the window is created. This way Latitude can make this communication to the gui layer and ensure that the window is created with a floating attribute.

In PowerPlant's UDesktop::NewDeskWindow() function, we want to inform Latitude that the window we're about to create is a floater. Just before the GetNewWindow() call at the bottom of this function, we insert the following:

```
#ifdef _LATITUDE_
  if (inWindow->HasAttribute(windAttr_Floating)) {
    LG_WMGR_WINDOW_BLOCK wblock;

    wblock.flags  = LG_WMGR_WINDOW_FLOAT;
    wblock.floats = LG_WINDOW_IS_FLOATER;
    lg_latitude_next_window(&wblock);
  }
#endif

  if  (UEnvironment::HasFeature(env_SupportsColor)) {
    macWindowP = ::GetNewCWindow(inWINDid, nil, inBehind);
  } else {
    macWindowP = ::GetNewWindow(inWINDid, nil, inBehind);
  }
```

CW Latitude has three settings for the floats attribute: LG_WINDOW_IS_FLOATER means that this window is to float above other windows that aren't specified to float. LG_WINDOW_HAS_FLOATERS is the default for regular windows. These windows allow others to float above them. Finally, LG_WINDOW_HAS_NO_FLOATERS specifies windows that must float above all else.

Controls As Drawings

On the Mac, standard controls are drawn in Quickdraw so the area they're rendered in, when scrolled, behaves as expected and the control scrolls with the area. This not the case with CW Latitude and the gui layers it supports. On systems like Motif, controls float above the window's canvas, so if drawings are scrolled across the window, the controls will not move by themselves. They must be explicitly moved.

PowerPlant's LPane::AdaptToSuperScroll() adjusts the control's rectangles but doesn't move the controls themselves. By overriding AdaptToSuperScroll for the LStdControl class and adding a Draw(nil) call, we can be sure that CW Latitude's gui layer will move the control for us.

Muscle

With as much PowerPlant as needed for Muscle ported, we can now begin looking at Muscle's source code. Since Muscle is almost entirely written in terms of PowerPlant, we have very little work to do.

In the ColorTutor example, we had to remove instances of qd. for CW Latitude. In C++ applications however, the QDGlobals type is a class of inline definitions tied to the Quickdraw low memory globals and the application must define it's own QDGlobals qd. So at the top of CMuscleApp.c, we add

```
#ifdef _LATITUDE_
QDGlobals qd;
#endif
```

We also add a lg_latitude_init() call similar to the one we added in ColorTutor.c. Muscle's signature is 'MePo'. We can also turn on CW Latitude's gui colorization so that windows have a default background color that matches the rest of their desktop.

```
void main( long argc, char **argv )
{
#ifdef _LATITUDE_
  LG_APP_INFO_BLOCK lblock;

  lblock.flags = LG_APP_INFO_SIGNATURE |
  LG_APP_INFO_CLASSNAME |
      LG_APP_INFO_WINDOW_GUI_COLOR;
  lblock.signature = QUADCONST( 'M','e','P','o' );
  lblock.classname = "Latitude";
  lblock.window_gui_color = 1;
  lg_latitude_init( argc, argv, &lblock );
#endif
```

Muscle contains a demonstration of PowerPlant's QuickTime support. Unfortunately, Rhapsody does not yet support QuickTime. Latitude will rely on Rhapsody's own QuickTime support once it becomes available. At this point, it is necessary for us to #ifdef out code involving QuickTime. This code is centralized in Muscle's CMuscleApp.cp file and is easy to spot. QuickTime initialization and destruction code in CMuscleApp::CMusleApp and CMuscleApp::~CMusleApp, QuickTime demo selection code in CMusleApp::ObeyCommand, and CMusleApp::FindCommandStatus can all be #ifdefed out.

Porting YOUR Application

As any seasoned Mac programmer will agree, there is a lot of lore in Mac programming. Having assisted many Mac porting projects using CodeWarrior Latitude since its beginnings, we have seen some porting efforts go smoothly and some go rough. Knowledge of Mac programming lore is key.

The make-up of the porting team makes all the difference in the world. The best team combines experience from both Mac and Rhapsody environments in addition to a strong understanding of the underlying Mac code base.

Less advantaged teams are made up entirely of either Mac programmers who aren't familiar with Rhapsody details or Rhapsody programmers who have no Mac programming experience and are not at all familiar with the code base being ported. Experience from both sides is necessary. Rhapsody and/or UNIX skills are needed to properly set up the development environment and address platform issues. Mac skills are needed to understand what exactly is being done in the application source code. By taking the time to put the right team together for the job, your porting project will be much smoother and easier.

We've touched on only a few aspects of the process of porting Mac applications to Rhapsody. Fortunately, PowerPlant and Muscle do not contain many of the *imaginative* coding techniques we've seen in other Mac applications. Many of these issues are discussed in the CodeWarrior Latitude Guide that accompanies the CodeWarrior Latitude software package.

We invite you to stop by Developer Central at MacWorld in San Francisco, January 6 - 9, 1998, and see Latitude in action. You can pick up the ColorTutor and other sample code at MacTech's web site. And don't forget to stop by www.metrowerks.com for up to the minute updates on CW Latitude and Metrowerks' other products.

PROGRAMMER'S CHALLENGE

by Bob Boonstra, Westford, MA

Cell Selection

One year ago, the Macintosh world was full of rumors about the prospect of Apple acquiring the Be Operating System. MacTech bundled the DR8 BeOS CD-ROM with the magazine. The Challenge column invited readers to try the BeOS on their Macs and enter the first BeOS Challenge. Then, before the BeOS issue had even arrived, Apple announced its intention to buy NeXT, and the journey to Rhapsody began.

Even though BeOS has faded from the headlines, it runs better on your Mac than it ever did. In fact, Preview Release 2 is scheduled to arrive in the immediate future. My personal interest in BeOS has been reinvigorated by a recent bargain that was too good to pass up – a dual 200MHz 604e MaxPOWR 400 processor upgrade. While it is possible for developers to take advantage of multiple processors under MacOS, using a special nonsymmetric interface developed in conjunction with DayStar, multiprocessing MacOS applications are the exception rather than the rule. Not so, of course, with BeOS, where symmetric multiprocessing is an intrinsic part of the operating system.

In honor of the one-year anniversary of the MacTech BeOS issue, and in celebration of my new 2x200 MHz toy, the Challenge this month is going to encourage the use of multiple processors. You will be able to use either BeOS or MacOS. If you choose to use MacOS and wish to take advantage of the second processor on my test system, you should use the SDK found at: <ftp://dev.apple.com/devworld/Development_Kits/Multiprocssing_SDK.sit.hqx>.

The problem this month, suggested by Jon "h++" Wätte, is to implement a CellSelection class. CellSelection implements a two-dimensional set of cells, each of which can be "on" or "off", along with a collection of methods to manipulate cell states.

The prototype for the code you should write is:

```
#if __dest_os == __be_os
#include <SupportDefs.h>
#else
typedef long int32;
typedef unsigned long uint32;
#endif

struct Area {
  int32    left,top,right,bottom;
    /* Area coordinates are inclusive. {2,2,3,4} includes 6 cells. */
    /* Any area with left>right or top>bottom is empty. */
};

class CellSelection {
  private:
    /* add your methods and instance variables here */
  public:
    CellSelection(void);
      /* create an empty selection */
    ~CellSelection(void);
      /* free any allocated memory */
    bool Clear();
      /* make the selection empty */
    bool Add(Area area);
      /* add the area of cells to this selection */
    bool Remove(Area area);
      /* remove the area of cells from this selection */
    bool Invert(Area area);
      /* remove cells in the area that are also in this selection
         and add the area cells that are not in this selection */
    bool Add(const CellSelection & otherSelection);
      /* add the otherSelection to this selection */
    bool Remove(const CellSelection & otherSelection);
      /* remove the otherSelection from this selection */
    bool Invert(const CellSelection & otherSelection);
      /* remove cells in the otherSelection that are also in this selection
         and add the otherSelection cells that are not in this selection */
    bool AllSelected(Area area);
      /* return TRUE if all cells in the area are selected */
    uint32 CountSelected(Area area);
      /* count cells that are "on" */
    bool EqualSelected(const CellSelection & otherSelection);
      /* return TRUE if otherSelection equals this selection */
};
```

THE RULES

Here's how it works: each month we present a new programming challenge. First, write some code that solves the challenge. Second, optimize your code (a lot). Then, submit your solution to MacTech Magazine. We choose a winner based on code correctness, speed, size, and elegance (in that order of importance) as well as the submission date. In the event of multiple equally desirable solutions, we'll choose one winner (with honorable mention, but no prize, given to the runner up). The prize for each month's best solution is a $100 credit for Developer Depot™.

Unless stated otherwise in the problem statement, the following rules apply: All solutions must be in ANSI compatible C or C++, or in Pascal. We disqualify entries with any assembly in them (except for challenges specifically stating otherwise.) You may call any Macintosh Toolbox routine (e.g., it doesn't matter if you use NewPtr instead of malloc). We compile all entries into native PowerPC code with compiler options set to enable all available speed optimizations. The development environment to be used for selecting the winner will be stated in the problem. **Limit your code to 60 characters per line** or compress and binhex the solution; this helps with e-mail gateways and page layout.

We publish the solution and winners for each month's Programmer's Challenge three months later. All submissions must be received by the 1st day of the month printed on the front cover of this issue.

You can get a head start on the Challenge by reading the Programmer's Challenge mailing list. It will be posted to the list on or before the 12th of the preceding month. To join, send an email to listserv@listmail.xplain.com with the subject "subscribe challenge-A".

Mark solutions "Attn: Programmer's Challenge Solution" and send it by e-mail to one of the Programmer's Challenge addresses in the "How to Communicate With Us" section on page 2 of this issue. Include the solution, all related files, and your contact info.

MacTech Magazine reserves the right to publish any solution entered in the Programmer's Challenge. Authors grant MacTech Magazine the exclusive right to publish entries without limitation upon submission of each entry. Authors retain copyrights for the code.

The destructor should free any memory allocated by the constructor or by any of the methods of CellSelection. The Add method should turn on any cells in the area or otherSelection; similarly the Remove method should turn off the specified cells. Invert is an exclusive-or operator that turns off cells that are on and turns on cells that are off, provided the cell is in the specified area or otherSelection. The Clear method resets the selection to its original empty state. All of these methods return TRUE if they succeed or FALSE if they run out of memory. The AllSelected method determines whether all of the cells in the specified area are on, while the CountSelected method counts the number of cells in the specified area that are on. Finally, the EqualSelected method determines whether the selected cells in this CellSelection are the same as the selected cells in the otherSelection.

The test code will require you to create a modest number of CellSelection instances (perhaps 10 to 50) and manipulate them with a larger number of Add/Remove/Invert operations, interspersed with a modest number of AllSelected/CountSelected/EqualSelected tests.

This will be a native PowerPC Challenge, using the latest CodeWarrior environment. Solutions must be coded in C++, written for either the Macintosh operating system or the Be operating system. You can use all of the available memory in my 96MB 8500, but your code must fail gracefully if it runs out of memory. Your solution should include a complete CodeWarrior project file and test driver, compressed into a .sit or .cpt archive (for MacOS) or into a .zip, .gz archive (for BeOS). I'll evaluate your solution using the target operating system you designate, and the fastest correct solution will be the winner. Memory-inefficient solutions that fail to handle large problems will be considered less correct than memory-efficient solutions.

Three Months Ago Winner

Congratulations to **Randy Boring** for submitting the winning entry to the Who Owns the Zebra Challenge. Recall that this Challenge required you to parse a set of inputs representing clues like "The American lives in the house with the red door" and "The person who drinks orange juice owns the dog" and solve a problem like "Who owns the zebra?" Randy's solution was faster than the second place solution in three of four test cases, and some 25% faster in the largest test case.

Randy's solution maintains a mask matrix, gLocations[], to indicate which locations are still possible for a (solution row, location) combination. He also maintains a value matrix gValue[] to indicate which location assignments remain possible for a given (variable, value) pair. Clues that constrain location assignments manipulate these data structures directly, while clues that relate (variable, value) pairs propagate the location constraints of each pair to the other pair in the routine ApplySAME_ROW. Constraint propagation is done in the ApplyXXX routines, which should be examined to understand how this solution works. If a solution is not evident when all of the clues have been applied, the ThinkRealHard routine hypothesizes a new constraint and propagates this artificial clue, terminating when a solution is reached or when it runs out of constraint hypotheses.

I would be remiss if I did not mention the Java solution submitted by David Whitney, in a self-professed "blatant violation" of the rules. Although his solution suffered in execution time, I found it interesting. Perhaps it is time for a Java Challenge... Any ideas?

The table below lists the execution times, code size, data size, and programming language for each entry. Execution time is listed in milliseconds for each of four problem dimension values tested: 3, 5, 15, and 31. Solutions which did not solve a test case in a reasonable amount of time (several minutes) are listed with an asterisk and did not win any Challenge points. The number in parentheses after the entrant's name is the total number of Challenge points earned in all Challenges to date prior to this one.

Name	Total Time 3	Time 5	Time 15	Time 31	Time	Code	Data	Lang.
Randy Boring (41)	728.6	0.3	0.6	31.3	696.4	8400	22530	C
Ernst Muter (300)	972.8	1.5	2.2	29.7	939.4	6420	128	C++
David Whitney	477247.0	134.0	410.0476703.0		*	45010	0	Java!
Willeke Rieken (10)	*	0.4	1.6	*	*	6920	548	C++
Ken Slezak (20)	*	0.4	*	*	*	6340	232	C

Top 20 Contestants

Here are the Top Contestants for the Programmer's Challenge. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	210	11.	Day, Mark	20
2.	Boring, Randy	61	12.	Higgins, Charles	20
3.	Cooper, Greg	61	13.	Larsson, Gustav	20
4.	Lewis, Peter	57	14.	Lengyel, Eric	20
5.	Nicolle, Ludovic	48	15.	Studer, Thomas	20
6.	Murphy, ACC	34	16.	Saxton, Tom	17
7.	Gregg, Xan	33	17.	Gundrum, Eric	15
8.	Mallett, Jeff	30	18.	Hart, Alan	14
9.	Antoniewicz, Andy	24	19.	O'Connor, Turlough	14
10.	Picao, Miguel Cruz	21	20.	Karsh, Bill	12

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place .....20 points
2nd place .....10 points
3rd place .....7 points
4th place.....4 points
5th place .....2 points
finding bug .....2 points
suggesting Challenge...2 points

Here is Randy's winning solution:

Zebra.c


```
#include "Zebra.h"

#include <string.h>
#include <stdio.h>
#include <stdlib.h>   // for malloc and free

typedef enum {
  kClueNextTo,
  kClueImmedRightOf,
  kClueImmedLeftOf,
  kClueSameRowAs,
  kClueLocatedAt
  } ZClueType;

typedef enum {
  kContradiction = -1,
  kSolved = 0,
  kUnsolved = 1
  } ZSolutionState;

// global string constants
// relations
static const CStr255 gsrISA = "ISA";   // variable membership
static const CStr255 gsrIS_LOCATED = "IS_LOCATED"; // absolute position
// relative position relations
static const CStr255 gsrNEXT_TO = "NEXT_TO";
static const CStr255 gsrIMMED_RIGHT_OF = "IMMED_RIGHT_OF";
static const CStr255 gsrIMMED_LEFT_OF  = "IMMED_LEFT_OF";
// absolute position values
static const CStr255 gsvIN_MIDDLE = "IN_MIDDLE";
static const CStr255 gsvAT_RIGHT = "AT_RIGHT";
static const CStr255 gsvAT_LEFT  = "AT_LEFT";
// parts of the Question
static const CStr255 gsqSOLVE = "SOLVE";
static const CStr255 gsqANSWER = "ANSWER";
static const long kMaxValues = 964;
static const long kMaxVariables = 31;

// bit array of possible locations
// has just a single bit set when location is known
typedef unsigned long ZLocation;

static const ZLocation kLeftBit = 0x80000000;
static ZLocation gMask;

typedef struct value {
  long       loc;        // index of possible locations for this value
  struct value  *next;   // next ZValue of this ZType
  void       *type;   // points to the variable (ZType) that we are a possible value of
  char       *name;  // name given for this ZValue
  } ZValueRec, *ZValue;

// a ZType is the definition of a variable, including its name and possible values
typedef struct type {
  ZValue       values;      // list of possible values for this variable
  struct type     *next;    // next variable of this problem
  char       *name;         // name given for this variable
  long       order;         // print order of this variable in solution
  } ZTypeRec, *ZType;

// a ZClue is a Fact that must be used to solve the puzzle
typedef struct clue {
  struct clue     *next;      // next clue in list
  ZClueType     type;         // which kind of clue
  ZValue       firstValue;    // first variable of this clue
  ZValue       secondValue;   // second variable of this clue
  } ZClueRec, *ZClue;

typedef struct q {
  ZClue     head;   // front of line where things are taken off
  ZClue     tail;   // back of line where things are added
  } ZQRec, *ZQ;

static unsigned long gDim;
static unsigned long gNumClues;
static unsigned long gkBlockSize;
static unsigned long gkBlockSizeInLongs;
static ZQRec gQRecUnsatisfied;
static ZQRec gQRecSatisfied;

static ZQ gQUn = &gQRecUnsatisfied;
static ZQ gQSat = &gQRecSatisfied;
static ZTypeRec gVars[kMaxVariables];    // 32 (31 is max)
static ZValueRec gValues[kMaxValues];    // (964) 31 X 31 (961) is max
static ZLocation gLocations[kMaxValues];// (964) 31 X 31 (961) is max
static ZLocation *gL = gLocations;


#define FirstVar()            (&(gVars[0]))
#define FirstValueOf(var)     ((var)->values)
#define NameOf(x)             ((x)->name)
#define SetNameOf(x, s)       ((x)->name = (s))
#define NextOf(x)             ((x)->next)
#define SetNextOf(x, y)       ((x)->next = (y))
#define LocationOf(v)         (gL[(v)->loc])
#define SetLocationOf(v, L)   (gL[(v)->loc] = (L))
#define TypeOf(c)             ((c)->type)
#define SetTypeOf(c, t)       ((c)->type = (t))
#define PrintOrderOf(var)     ((var)->order)
#define SetPrintOrder(var,n)  ((var)->order = (n))
#define FirstValOf(c)         ((c)->firstValue)
#define SetFirstValOf(c, v)   ((c)->firstValue = (v))
#define SecondValOf(c)        ((c)->secondValue)
#define SetSecondValOf(c, v)  ((c)->secondValue = (v))
```

InitQ

```
// Initialize queue q
static void
InitQ(ZQ q)
{
  q->head = nil;
  q->tail = nil;
}


#define QNotEmpty(q) ((q)->head != nil)
#define QIsEmpty(q) ((q)->head == nil)

// Add c to end of queue q
static void
EnQ(ZQ q, ZClue c)
{
  if (q->head == nil)
    q->head = c;
  else
    SetNextOf(q->tail, c);
  q->tail = c;
  SetNextOf(c, nil);
}

// Remove the clue at the front of queue q and Return it
static ZClue
DeQ(ZQ q)
{
  ZClue c = q->head;
  q->head = NextOf(c);
  SetNextOf(c, nil);
  if (q->head == nil)
    q->tail = nil;
  return c;
}

// Add the elements of the first queue to the end of the
// second queue (removing them from the first one)
static void
MergeQ(ZQ from, ZQ to)
{
  while (QNotEmpty(from))
    EnQ(to, DeQ(from));
}

static void
MoveQ(ZQ from, ZQ to)
{
  to->head = from->head;
  to->tail = from->tail;
}

static void
MakeMask(long dim)
{
  ZLocation currBit = kLeftBit;
  gMask = 0;
  while (dim--)
    {
```

```
    gMask = gMask | currBit;
    currBit >>= 1;
    }
}
```

AddLocations

```
// Make the variable field for the problem
// dim rows, each with dim types
static void
AddLocations(long dim)
{
  long vari, vali = 0, loci = 0;
  for (vari = 0; vari < dim; vari++)
    {
    long addValuesStop = vali + dim;
    gVars[vari].values = &(gValues[vali]);
    gVars[vari].next = &(gVars[vari + 1]);
    gVars[vari].name = nil;
    do  {
      gL[loci] = gMask;
      gValues[vali].loc = loci++;
      gValues[vali].next = &(gValues[vali + 1]);
      gValues[vali].type = &(gVars[vari]);
      gValues[vali].name = nil;
      vali++;
      } while (vali < addValuesStop);
    gValues[vali - 1].next = nil;  // terminate the list
    }
  gVars[vari - 1].next = nil;  // terminate the list
}
```

InitStructure

```
static void
InitStructure(long numClues, long problemDimension)
{
  gkBlockSizeInLongs = problemDimension * problemDimension;
  gDim = problemDimension;
  gNumClues = numClues;
  gkBlockSize = gkBlockSizeInLongs * sizeof(ZLocation);
  MakeMask(problemDimension);
  AddLocations(gDim);
  InitQ(gQUn);
  InitQ(gQSat);
}

static void
ReleaseMemory(void)
{
  while (QNotEmpty(gQUn))
    free(DeQ(gQUn));
  while (QNotEmpty(gQSat))
    free(DeQ(gQSat));
}

static void
CopyLocations(ZLocation *from, ZLocation *to)
{
  BlockMoveData((Ptr) from, (Ptr) to, gkBlockSize);
}

// Starting at a word beginning, search for its end.
// Null-terminate the word, null-out any extra space, and
// return the location of the next word.
static char *
MakeWordBreak(char *w)
{
  while (*w != ' ' && *w)
    w++;
  while (*w == ' ')
    *w++ = 0;
  return w;
}

static ZType
FindOrCreateType(char *aVariable)
{
  ZType t = FirstVar();
  while (t != nil && NameOf(t) != nil
      && (0 != strcmp(NameOf(t), aVariable)))
    t = NextOf(t);
  if (t != nil)
    SetNameOf(t, aVariable);
  return t;
}
```

```
static void
AddPossibleValue(ZType var, char *aValue)
{
  ZValue v = FirstValueOf(var);
  while (v != nil && NameOf(v) != nil
      && (0 != strcmp(NameOf(v), aValue)))
    v = NextOf(v);
  if (v != nil)
    SetNameOf(v, aValue);
  else
    DebugStr("\pv is nil!");
}

static ZLocation
FindLocation(char *aLocation)
{
  ZLocation loc = 0;
  if (strcmp(aLocation, gsvAT_LEFT) == 0)
    loc = kLeftBit;
  else if (strcmp(aLocation, gsvAT_RIGHT) == 0)
    loc = kLeftBit >> (gDim - 1);
  else if (strcmp(aLocation, gsvIN_MIDDLE) == 0)
    loc = kLeftBit >> (gDim >> 1);
  else
    DebugStr("\p Not a valid location!");
  return loc;
}

static ZType
FindTypeNamed(char *aVariable)
{
  ZType t = FirstVar();
  while (t != nil && (0 != strcmp(NameOf(t), aVariable)))
    t = NextOf(t);
  return t;
}

static ZType
FindTypeOrdered(long n)
{
  ZType t = FirstVar();
  while (t != nil && PrintOrderOf(t) != n)
    t = NextOf(t);
  return t;
}
```

FindValueNamed

```
// Return the ZValue that is named aValue
static ZValue
FindValueNamed(char *aValue)
{
  ZType var;
  ZValue v = nil;
  for (var = FirstVar(); var != nil; var = NextOf(var))
    {
    v = FirstValueOf(var);
    while (v != nil && (0 != strcmp(NameOf(v), aValue)))
      v = NextOf(v);
    if (v != nil)
      return v;
    }
  return v;
}
```

IsSolvedLoc

```
// Returns true if the location, n, is 'solved', that is,
// if it has exactly one bit set.
static ZSolutionState
IsSolvedLoc(ZLocation n)
{
  // no set bits found yet
  ZLocation currBit = kLeftBit;
  long i = gDim;
  if (0 == n)
    {
//  DebugStr("\p Oversolved bit!");
    return kContradiction;
    }
  while (i--)
    {
    if (currBit & n)
      break;
    currBit >>= 1;
    }
  currBit >>= 1;
  // one set bit found
  while (i--)
    {
    if (currBit & n)
      return kUnsolved;    // two set bits found!
    currBit >>= 1;
    }
  return kSolved;    // exactly one set bit was found
}
```

FindValueWithUnsolvedLocation

```
// Return the first ZValue that is has an unsolved location
// That is, the location is not just a single bit.
static ZValue
FindValueWithUnsolvedLocation(void)
{
  ZType var;
  ZValue v = nil;
  for (var = FirstVar(); var != nil; var = NextOf(var))
    {
    v = FirstValueOf(var);
    while (v != nil && (kSolved ==
IsSolvedLoc(LocationOf(v))))
      v = NextOf(v);
    if (v != nil)
      return v;
    }
  return v;
}

// NOTE: watch out for the typecast in here
static void
AddFactLocationAbsolute(ZLocation loc, ZValue val)
{
  ZClue c = malloc(sizeof(ZClueRec));
  SetTypeOf(c, kClueLocatedAt);
  SetFirstValOf(c, val);
  SetSecondValOf(c, (ZValue) loc); // overload!
  EnQ(gQUn, c);
}

static void
AddFactLocationRelativeNextTo(ZValue aVal, ZValue bVal)
{
  ZClue c = malloc(sizeof(ZClueRec));
  SetTypeOf(c, kClueNextTo);
  SetFirstValOf(c, aVal);
  SetSecondValOf(c, bVal);
  EnQ(gQUn, c);
}

static void
AddFactLocationRelativeOnRight(ZValue aVal, ZValue bVal)
{
  ZClue c = malloc(sizeof(ZClueRec));
  SetTypeOf(c, kClueImmedRightOf);
  SetFirstValOf(c, aVal);
  SetSecondValOf(c, bVal);
  EnQ(gQUn, c);
}

static void
AddFactLocationRelativeOnLeft(ZValue aVal, ZValue bVal)
{
  ZClue c = malloc(sizeof(ZClueRec));
  SetTypeOf(c, kClueImmedLeftOf);
  SetFirstValOf(c, aVal);
  SetSecondValOf(c, bVal);
  EnQ(gQUn, c);
}

static void
AddFactSameRow(ZValue aVal, ZValue bVal)
{
  ZClue c = malloc(sizeof(ZClueRec));
  SetTypeOf(c, kClueSameRowAs);
  SetFirstValOf(c, aVal);
  SetSecondValOf(c, bVal);
  EnQ(gQUn, c);
```

```
}

static void
AddFactImportantValue(ZValue val)
{
#pragma unused (val)
}

static void
AddFactImportantType(ZType type)
{
#pragma unused (type)
}
```

ProcessISA

```
// Associates aValue with aVariable, so that we know that
//  aValue is a possible value for the ZType aVariable.
// The ZType aVariable is created if it doesn't already exist.
static void
ProcessISA(char *aValue, char *aVariable)
{
  ZType v;
  v = FindOrCreateType(aVariable);
  AddPossibleValue(v, aValue);
}
```

ProcessIS_LOCATED

```
// Associates aValue with aLocation, so that we know that
//  aValue is at the location aLocation.
static void
ProcessIS_LOCATED(char *aValue, char *aLocation)
{
  ZLocation loc;
  ZValue val;

  loc = FindLocation(aLocation);
  val = FindValueNamed(aValue);
  AddFactLocationAbsolute(loc, val);
}
```

ProcessNEXT_TO

```
// Associates aValue with bValue, so that we know that  aValue is located next to
// bValue. (This also implies that bValue is next to aValue)
static void
ProcessNEXT_TO(char *aValue, char *bValue)
{
  ZValue a, b;
  a = FindValueNamed(aValue);
  b = FindValueNamed(bValue);
  AddFactLocationRelativeNextTo(a, b);
}
```

ProcessIMMED_RIGHT_OF

```
// Associates aValue with bValue, so that we know that aValue is located to the
// immediate right of bValue. (Also, bValue is to the immediate left of aValue)
static void
ProcessIMMED_RIGHT_OF(char *aValue, char *bValue)
{
  ZValue a, b;
  a = FindValueNamed(aValue);
  b = FindValueNamed(bValue);
  AddFactLocationRelativeOnRight(a, b);
}
```

ProcessIMMED_LEFT_OF

```
// Associates aValue with bValue, so that we know that aValue is located to the
// immediate left of bValue. (Also, bValue is to the immediate right of aValue)
static void
ProcessIMMED_LEFT_OF(char *aValue, char *bValue)
{
  ZValue a, b;
  a = FindValueNamed(aValue);
  b = FindValueNamed(bValue);
  AddFactLocationRelativeOnLeft(a, b);
}
static void
DoubleNullTerminate(char *line)
{
  long len = strlen(line);
  line[len + 1] = 0;
}
```

ProcessSOLVE

```
// Remembers that the remaining words are important, both values and
// variables. The solution is done when every value mentioned here is known
// for certain, and every variable has a compatible (if not guaranteed) value.
static void
ProcessSOLVE(char *currWord, char *nextWord)
{
```

```
  DoubleNullTerminate(nextWord);
  while (*currWord)
    {
    ZValue val = FindValueNamed(currWord);
    if (val)
      AddFactImportantValue(val);
    else
      {
      ZType type = FindTypeNamed(currWord);
      if (type)
        AddFactImportantType(type);
  // else it was a relation
      }
    currWord = nextWord;
    nextWord = MakeWordBreak(nextWord);
    }
}
```

ProcessANSWER

```
// Remembers the ordering of the remaining words.
//This is the order that the fields in the solution lines
// are filled with variable values. (zero-based)
static void
ProcessANSWER(char *currWord, char *nextWord)
{
  long i = 0;
  ZType type;
  DoubleNullTerminate(nextWord);
  while (*currWord)
    {
    type = FindTypeNamed(currWord);
    if (!type)
      DebugStr("\pCan't find this type!");
    else
      {
      SetPrintOrder(type, i);
      i++;
      }
    currWord = nextWord;
    nextWord = MakeWordBreak(nextWord);
    }
}
```

ProcessRelation

```
static void
ProcessRelation(char *firstWord, char *relation, char *thirdWord)
{
#pragma unused (relation)
  ZValue aVal;
  ZValue bVal;
  aVal = FindValueNamed(firstWord);
  if (!aVal)
    return;  // this is probably a meaningless 'variable relation variable' clue
  bVal = FindValueNamed(thirdWord);
  if (!bVal)
    {
    DebugStr("\pNot a known value!");
    return;  // this is probably an error!
    }
  AddFactSameRow(aVal, bVal);
}
```

ProcessAClue

```
static void
ProcessAClue(CStr255 clue)
{
  char *firstWord = clue, *secondWord, *nextWord;
  secondWord = MakeWordBreak(firstWord);
  nextWord = MakeWordBreak(secondWord);
  if (0 == strcmp(gsrISA, secondWord))
    ProcessISA(firstWord, nextWord);
  else if (0 == strcmp(gsrIS_LOCATED, secondWord))
    ProcessIS_LOCATED(firstWord, nextWord);
  else if (0 == strcmp(gsrNEXT_TO, secondWord))
    ProcessNEXT_TO(firstWord, nextWord);
  else if (0 == strcmp(gsrIMMED_RIGHT_OF, secondWord))
    ProcessIMMED_RIGHT_OF(firstWord, nextWord);
  else if (0 == strcmp(gsrIMMED_LEFT_OF, secondWord))
    ProcessIMMED_LEFT_OF(firstWord, nextWord);
  else if (0 == strcmp(gsqSOLVE, firstWord))
    ProcessSOLVE(secondWord, nextWord);
  else if (0 == strcmp(gsqANSWER, firstWord))
    ProcessANSWER(secondWord, nextWord);
  else
    ProcessRelation(firstWord, secondWord, nextWord);
}

void PropogateUniqueLocInColumn(ZValue val);
```

ClearColumn

```
// Clear this bit in every value in this variable Except 'skip' (the
// source of clearing the others) Propogate newly solved locations,
// too. Don't propogate contradictions (zeroes)
static void
ClearColumn(ZType column, ZLocation bit, ZValue skip)
{
  const ZLocation clearMask = ~bit;
  ZValue v;
  for (v = FirstValueOf(column); v != nil; v = NextOf(v))
    if (v == skip)
      continue;
    else {
      ZLocation originalLoc = LocationOf(v);
      ZLocation newLoc = originalLoc & clearMask;
      if (newLoc != originalLoc)
        {
        ZSolutionState st = IsSolvedLoc(newLoc);
        SetLocationOf(v, newLoc);
        if (newLoc && st == kSolved)
          PropogateUniqueLocInColumn(v);
    //else if (st == kContradiction)
    //else if (st == KUnsolved)
        }
      }
}
```

PropogateUniqueLocInColumn

```
// Erases this value's location bit from every other value
// of this variable/column/type
static void
PropogateUniqueLocInColumn(ZValue val)
{
  ZLocation loc = LocationOf(val);
  SetLocationOf(val, loc);
  ClearColumn((ZType)TypeOf(val), loc, val);
}

typedef enum {
  kResultContradiction,
  kResultProgress,
  kResultSolvedFirst,
  kResultSolvedSecond,
  kResultSolvedBoth,
  kResultNoProgress
  } ZApplyResult;
```

CalculateApplyResult

```
static ZApplyResult
CalculateApplyResult(ZLocation result1, ZLocation result2,
    ZLocation original1, ZLocation original2)
{
  if (!result1 || !result2)
    return kResultContradiction;
  if (result1 != original1)
    if (IsSolvedLoc(result1) == kSolved)
      if (result2 != original2)
        if (IsSolvedLoc(result2) == kSolved)
          return kResultSolvedBoth;
        else
          return kResultSolvedFirst; // and progress on 2nd
      else
        return kResultSolvedFirst;
    else // progress on first
      if (result2 != original2)
        if (IsSolvedLoc(result2) == kSolved)
          return kResultSolvedSecond; // and progress on 1st
        else
          return kResultProgress;
      else
        return kResultProgress;
  else if (result2 != original2)
    if (IsSolvedLoc(result2) == kSolved)
      return kResultSolvedSecond;
    else
      return kResultProgress;
  else
    return kResultNoProgress;
}
```

ApplyNEXT_TO

```
static ZApplyResult
ApplyNEXT_TO(ZValue first, ZValue second)
{
  ZLocation original1, original2;
  ZLocation result1 = original1 = LocationOf(first);
  ZLocation result2 = original2 = LocationOf(second);
  ZLocation temp1, temp2;
  do  {
    temp1 = result1;
    temp2 = result2;
  // first is to right or left of second
    result1 &= ((temp2 << 1) | (temp2 >> 1));
  // second is to right or left of first
    result2 &= ((temp1 << 1) | (temp1 >> 1));
    } while ((temp1 != result1) || (temp2 != result2));
  SetLocationOf(first, result1);
  SetLocationOf(second, result2);
  return CalculateApplyResult(result1, result2,
          original1, original2);
}
```

ApplyRIGHT_OF

```
static ZApplyResult
ApplyRIGHT_OF(ZValue first, ZValue second)
{
  ZLocation original1, original2;
  ZLocation result1 = original1 = LocationOf(first);
  ZLocation result2 = original2 = LocationOf(second);
  ZLocation temp1, temp2;
  do  {
    temp1 = result1;
    temp2 = result2;
  // first is to right of second
    result1 &= (temp2 >> 1);
  // second is to left of first
    result2 &= (temp1 << 1);
    } while ((temp1 != result1) || (temp2 != result2));
  SetLocationOf(first, result1);
  SetLocationOf(second, result2);
  return CalculateApplyResult(result1, result2,
          original1, original2);
}
```

ApplyLEFT_OF

```
static ZApplyResult
ApplyLEFT_OF(ZValue first, ZValue second)
{
  ZLocation original1, original2;
  ZLocation result1 = original1 = LocationOf(first);
  ZLocation result2 = original2 = LocationOf(second);
  ZLocation temp1, temp2;
  do  {
    temp1 = result1;
    temp2 = result2;
  // first is to left of second
    result1 &= (temp2 << 1);
  // second is to right of first
    result2 &= (temp1 >> 1);
    } while ((temp1 != result1) || (temp2 != result2));
  SetLocationOf(first, result1);
  SetLocationOf(second, result2);
  return CalculateApplyResult(result1, result2,
          original1, original2);
}
```

ApplySAME_ROW

```
static ZApplyResult
ApplySAME_ROW(ZValue first, ZValue second)
{
  ZLocation original1 = LocationOf(first);
  ZLocation original2 = LocationOf(second);
 // first is in the same row as the second
  ZLocation result = (original1 & original2);
  SetLocationOf(first, result);
  SetLocationOf(second, result);
  return CalculateApplyResult(result, result,
          original1, original2);
}
```


ApplyLOCATED_AT

```
// Set this values location to be loc
// NOTE: Clearing this bit in every other value in this variable/column happens later
static ZApplyResult
ApplyLOCATED_AT(ZValue first, ZLocation loc)
{
 // first is located at loc
  SetLocationOf(first, loc);
  return kResultSolvedFirst;
}
```

ApplyAClue

```
static ZApplyResult
ApplyAClue(ZClue clue)
{
  ZValue first = FirstValOf(clue);
  ZValue second = SecondValOf(clue);
  ZApplyResult rslt;
  switch(TypeOf(clue))
    {
    case kClueNextTo: rslt =
      ApplyNEXT_TO(first, second);
      break;
    case kClueImmedRightOf: rslt =
      ApplyRIGHT_OF(first, second);
      break;
    case kClueImmedLeftOf: rslt =
      ApplyLEFT_OF(first, second);
      break;
    case kClueSameRowAs: rslt =
      ApplySAME_ROW(first, second);
      break;
    case kClueLocatedAt: rslt =
      ApplyLOCATED_AT(first, (ZLocation) second);
      break;
    }
  switch (rslt)
    {
    case kResultSolvedBoth:
```

```
      PropogateUniqueLocInColumn(first);
      // fall through for second, too
    case kResultSolvedSecond:
      PropogateUniqueLocInColumn(second);
      break;
    case kResultSolvedFirst:
      PropogateUniqueLocInColumn(first);
      break;
    default:
    case kResultContradiction:
    case kResultProgress:
    case kResultNoProgress:
      break;
    }
  return rslt;
}
```

IsSatisfied

```
// Returns true if the clue has been satisfied
// ASSUMES it has just been applied (thus several check only one value)
static ZSolutionState
IsSatisfied(ZClue clue)
{
  ZSolutionState satisfied;
  switch(TypeOf(clue))
    {
    case kClueNextTo: satisfied =
      IsSolvedLoc(LocationOf(FirstValOf(clue)));
      if (satisfied == kSolved)
        satisfied =
          IsSolvedLoc(LocationOf(SecondValOf(clue)));
      break;
    case kClueImmedRightOf:
    case kClueImmedLeftOf:
    case kClueSameRowAs:
    case kClueLocatedAt: satisfied =
      IsSolvedLoc(LocationOf(FirstValOf(clue)));
      break;
    }
  return satisfied;
}
```

CheckLocations

```
// Check one column, col, for newly unique locations. Check only the locations
// left in val. Check only the values below/after val, since the others were 'solved' (had
// unique solutions already) Stop after finding the first one (and making it unique)
// Return whether any changes were made
static Boolean
CheckLocations(ZType col, ZValue val)
{
  ZLocation currBit = kLeftBit;
  const ZLocation tryLocs = LocationOf(val);
  while (gMask & currBit)   // for each location in puzzle
    {
    ZValue tryv;
    if (currBit & tryLocs != 0) // just locations in val
      {
      // check remaining values for this location
      for (tryv = NextOf(val); tryv != nil; tryv = NextOf(tryv))
        if (LocationOf(tryv) & currBit)
          break;    // found another value that could be in this location
      if (!tryv)    // a unique location!
        {
        SetLocationOf(val, currBit);
        ClearColumn(col, currBit, val);
        return true;
        }
      }
    currBit >>= 1;
    }
  return false;
}
```

CheckColumns

```
// Check each column for newly unique locations
// Return whether any changes were made
static Boolean
CheckColumns(void)
{
  ZType var;
  ZValue v = nil;
  Boolean changes = false;
  for (var = FirstVar(); var != nil; var = NextOf(var))
    {
    v = FirstValueOf(var);
    while (v != nil && (kSolved == IsSolvedLoc(LocationOf(v))))
      v = NextOf(v);
    if (v != nil)
      changes = changes || CheckLocations(var, v);
    }
  return changes;
}
```

ApplyEachUnsatisfiedClue

```
// Apply each clue in the 'unsatisfied' queue. Return what progress was made, unless
// a contradiction was made, then return false to stop. Move satisfied clues to qSat.
static Boolean
ApplyEachUnsatisfiedClue(ZQ qSat)
{
  Boolean progress = false;
  ZQRec stillUnsatisfied;
  InitQ(&stillUnsatisfied);
  while (QNotEmpty(gQUn))
    {
    ZClue clue = DeQ(gQUn);
    ZApplyResult rslt = ApplyAClue(clue);
    if (rslt == kResultContradiction)
      {
      EnQ(qSat, clue);
      MergeQ(&stillUnsatisfied, gQUn);
      return false;   // contradiction reached
      }
    if ((rslt == kResultSolvedBoth) ||
      (rslt == kResultSolvedFirst &&
        TypeOf(clue) == kClueLocatedAt))
      {
      progress = true;
      EnQ(qSat, clue);
      }
    else if (rslt == kResultNoProgress)
      EnQ(&stillUnsatisfied, clue);
    else {
      progress = true;
      EnQ(&stillUnsatisfied, clue);
      }
    }
  MoveQ(&stillUnsatisfied, gQUn);
  if (!progress)
    progress = CheckColumns();
  return progress;
}
```

VerifyCluesStillSatisfied

```
// Verify each clue in the queue. Return true if no clues fail to say they are satisfied
static Boolean
VerifyCluesStillSatisfied(ZQ q)
{
#pragma unused(q)
  return true;
}
```

VerifyNoContradictions

```
// Verify each location in the current block, gL
// Return true if no locations are zero (i.e., impossible)
static Boolean
VerifyNoContradictions(void)
{
  ZLocation *currLoc = gL;
  ZLocation *stop = gL + gkBlockSizeInLongs;
  while (currLoc < stop)
    if (*currLoc++ == 0)
      return false;
  return true;
}
```

WriteRow

```
// 'loc' has just the bit set in the row position that we're looking for
static void
WriteRow(CStr255 row, ZLocation loc)
{
  ZValue v;
  long i;
  row[0] = 0;
  for(i = 0; i < gDim; i++)
    {
    ZType t = FindTypeOrdered(i);
    for (v = FirstValueOf(t); v != nil; v = NextOf(v))
```

```
    if ((LocationOf(v) & loc) != 0)
      {  // add the value's name to the line
      long len;
      strcat(row, NameOf(v));
      len = strlen(row);
      row[len] = 0x20;      // space between words
      row[len+1] = 0;       // re-null-terminate
      }
  }
}
```

WriteDownProblem

```
static void
WriteDownProblem(CStr255 clues[], long numClues, long dim)
{
  long i;
  InitStructure(numClues, dim);
  for (i = 0; i < numClues; i ++)
    ProcessAClue(clues[i]);
}
```

ThinkRealHard

```
// Apply clues until no more progress can be made, then enumerate all
// remaining possible configurations and test them recursively. Set the top-level
// location block to the first configuration that does not contradict any facts.
static Boolean
ThinkRealHard(ZLocation *myBlock)
{
  ZQRec newlySatisfied;
  ZQ qNew = &newlySatisfied;
  ZValue vtry;
  ZLocation trylocs;
  ZLocation currBit;
  long i;
  Boolean progressMade;
  InitQ(qNew);
  // Apply knowledge
  do  progressMade = ApplyEachUnsatisfiedClue(qNew);
  while (progressMade);
  // Test all locations for impossibility (zero possible
  // locations for a value means there was a contradiction)
  if (false == VerifyNoContradictions())
    {  // we have reached a contradiction
       // put the newly 'satisfied' clues back in the
       // unsatisfied list
    MergeQ(qNew, gQUn);
    return false;
    }
  // Generate-and-Test
  vtry = FindValueWithUnsolvedLocation();
  if (!vtry)
    {  // we have no more unsolved locations; done!
    MergeQ(qNew, gQUn);  // put them back so they can be freed
    return true;         // might be a solution
    }
  trylocs = LocationOf(vtry);
  currBit = kLeftBit;
  for (i = 0; i < gDim; i++, currBit >>= 1)
    if (currBit & trylocs)
      {
      ZLocation newBlock[kMaxValues];
      CopyLocations(myBlock, newBlock);
      gL = newBlock;
      // try new configuration by faking a clue
      ApplyLOCATED_AT(vtry, currBit);
      PropogateUniqueLocInColumn(vtry);
      if (true == ThinkRealHard(newBlock))
        {     // this configuration worked
        gL = myBlock;
        if (VerifyCluesStillSatisfied(qNew))
          {  // and verifies, so far, up to this level
          CopyLocations(newBlock, myBlock);
```

```
            MergeQ(qNew, gQUn); // put them back so they can be freed
            return true;        // looks like a solution
            }
          MergeQ(qNew, gQUn);   // put them back so they can be freed
          return false;         // failed verify
          }
        gL = myBlock;
        }
    MergeQ(qNew, gQUn);
    return false; // exhausted all configurations, none worked
}
```

WriteOutSolution

```
static void
WriteOutSolution(CStr255 solution[])
{
  long i = 0;
  for (i = 0; i < gDim; i++)
    WriteRow(solution[i], kLeftBit >> i);
  ReleaseMemory();
}
```

WhoOwnsZebra

```
void WhoOwnsZebra(
  long problemDimension, /* number of problem variables */
  long numClues,         /* number of clues provided */
  CStr255 clues[],       /* the clues */
  CStr255 solution[]     /* storage for problemDimension result strings */
) {
  // Dr. Richard Feynmann's Problem-Solving Algorithm:
  WriteDownProblem(clues, numClues, problemDimension);
  ThinkRealHard(gL);
  WriteOutSolution(solution);
}
```

MT

Room for Development

APS Technologies carries everything you need to store, archive, protect, backup, transport and distribute your next killer app. Listed below are just a few of our most popular products, give us a call to order a product, call for a free APS catalog; call for information... just call 1-800-761-8133. Or visit our online catalog, at www.apstech.com for literally thousands of storage solutions, peripherals and supplies. If you work in a mixed-platform environment, we've still got you covered – with Mac, PC and NT products.

APS HIGH-PERFORMANCE ULTRA SCSI DRIVES

Model	Description	EL	Int.	SR 2000	Pro
APS Q 2000	Quantum Fireball ST, 2079MB, 5.4K	$219.95	$229.95	$299.95	$329.95
APS Q 3000	Quantum Fireball ST, 3118MB, 5.4K	289.95	299.95	369.95	399.95
APS Q 4000	Quantum Fireball ST, 4136MB, 5.4K	319.95	339.95	409.95	439.95
APS Q 6400	Quantum Fireball ST, 6236MB, 5.4K	429.95	449.95	519.95	549.95
APS Q 4500	Quantum Viking, 4345MB, 7200 rpm		599.95	669.95	699.95
APS Q 4300	Quantum Atlas II, 4341MB, 7200 rpm		599.95	669.95	699.95
APS Q 9000	Quantum Atlas II, 8682MB, 7200 rpm		879.95	949.95	979.95
APS ST 2000	Seagate 32272N, 2157MB, 7200 rpm		Call	449.95	479.95
APS ST 4300	Seagate Barracuda, 4340MB, 7200 rpm		Call	649.95	679.95
APS ST 4500	Seagate Cheetah, 4348MB, 10000 rpm		N/A	N/A	779.95
APS ST 9000	Seagate Barracuda, 8683MB, 7200 rpm		Call	1,019.95	1,049.95
APS ST 9100	Seagate Cheetah, 8681MB, 10000 rpm		N/A	N/A	1,199.95
APS WD 2000	Western Digital Enterprise, 7200 rpm		399.95	469.95	499.95
APS WD 4300	Western Digital Enterprise, 7200 rpm		599.95	669.95	699.95

APS HIGH-PERFORMANCE ULTRA SCSI DRIVES

Model	Description	Full Height
APS ST 23000	Seagate Elite 23, 22.1GB, 5400 rpm	$1,999.95
APS ST 23000 W	Seagate Elite 23, 22.1GB, 5400 rpm	2,099.95

APS ULTRA WIDE SCSI DRIVES

Model	Description	Int.	SR 2000	Pro
APS Q 4300 W	Quantum Atlas II, 4341MB, 7200 rpm	$619.95	$689.95	$719.95
APS Q 4500 W	Quantum Viking, 4345MB, 7200 rpm	619.95	689.95	719.95
APS Q 9000 W	Quantum Atlas II, 8682MB, 7200 rpm	899.95	969.95	999.95
APS ST 2000 W	Seagate 82272W, 2157MB, 7200 rpm	Call	499.95	529.95
APS ST 4300 W	Seagate Barracuda, 4340MB, 7200 rpm	Call	699.95	729.95
APS ST 4500 W	Seagate Cheetah, 4348MB, 10,000 rpm	N/A	N/A	829.95
APS ST 9000 W	Seagate Barracuda, 8683MB, 7200 rpm	Call	1,069.95	1,099.95
APS ST 9100 W	Seagate Cheetah, 8681MB, 10,000 rpm	N/A	N/A	1,249.95
APS WD 2000 W	Western Digital Enterprise, 7200 rpm	429.95	499.95	529.95
APS WD 4300 W	Western Digital Enterprise, 7200 rpm	629.95	699.95	729.95

APS IDE DRIVES

Model	Description	Internal
APS Q 3000	Quantum Fireball ST, 3118MB, 5400 rpm	$229.95
APS Q 4000	Quantum Fireball ST, 4136MB, 5400 rpm	279.95
APS Q 6400	Quantum Fireball ST, 6236MB, 5400 rpm	389.95

APS RAID

Model	Description	Internal
APS ShortStack	Quantum Ultra SCSI 6GB/12GB	$1,299.95
APS Hardware RAID	Quantum Ultra SCSI w/5 - 4.3GB	7,999.95

APS POWERBOOK STORAGE

Model	Description	Internal
APS PowerBook Drive	IBM DMAA-21080, 1080MB, 4000 rpm	$399.95

APS REMOVABLE DRIVES

Model	Description	Int.	SR 1000	SR 2000
APS SQ 5200	SyQuest 5200, 190MB	N/A	N/A	$389.95
APS Jaz	(with 1 cartridge) 1GB	N/A	$399.95	399.95

APS MO DRIVES

Model	Description	SR 1000	SR 2000	Pro
APS 230 MO	(with 1 cartridge) 217MB	$299.95	$369.95	$399.95
APS 640 MO	Fujitsu M2513A2#N	N/A	469.95	499.95
APS 2.6GB MO	Sony SMO-F544, 2.4GB	N/A	1,769.95	1,799.95

APS CD-ROM DRIVES

Model	Description	External
APS CD24	24X CD-ROM in Slimline Case	$179.95
APS CD-R Plus	2X record/6X read CD-R (Sony)	379.95
APS CD-R Pro	4X record/6X read CD-R in Pro Enclosure	529.95
APS CD-RW	2X record/6X read CD-RW in Pro Enclosure	529.95
APS Jaz/CD-R	System 2X record/6X read CD-R	799.95
APS 2GB/CD-RW	Sys.Q 2000 Hard Drive & 2X record /6X read CD-RW	899.95
APS CD-R Pro	4X record/6X read CD-R in Full Height Enclosure	729.95

‡Available in an SR 2000 Enclosure for an additional $100.00

APS TAPE BACKUP SYSTEMS

Model	Description	Internal	External
APS HyperQIC®	Travan 4 Conner QIC 3095, 8GF	$349.95	$399.95
APS HyperDAT®	DDS-2DC, 8GB	749.95	799.95
APS HyperDAT® Pro	DDS-2DC, 8GB	849.95	899.95
APS HyperDAT® III	DDS-3DC, 24GB	1,149.95	1,199.95
APS Mini Library	4mm DDS-2 AutoLoader, 64GB	N/A	2,399.95
APS DLT40	DLT 4000, 40GB	N/A	2,999.95
APS DLT70	DLT 7000, 70GB	N/A	6,999.95
APS AIT*	8mm W/DC, 50GB	N/A	3,199.95
APS DL1	4MM DDS-2 Autoloader, 136GB *Now Even Faster!*	N/A	4,999.95

Visit Our Online Catalog at www.apstech.com

- *30-day money-back satisfaction on all APS brand drives and accessories. Your risk is the cost of shipping.*
- *SCSI cables sold separately.*
- *Drive-for-Drive Repair or Replacement Warranty. APS will, at its discretion, replace or repair products found to be defective according to the terms of the product's warranty.*
- *Refused orders subject to 20% restocking fee.*
- *International customers must pay for all shipping charges.*
- *Listed capacities are formatted.*
- *Actual data compression and tape capacity vary greatly depending on the type of data recorded, other system parameters and environment.*
- *Prices and specifications are subject to change without notice.*
- *You need to install system software appropriate to your machine before using our hard drives.*
- *Not responsible for typographical errors.*

1-800-761-8133

APS Technologies

APS Technologies
6131 Deramus
Kansas City, MO 64120

Developer Central™

"The Place for Macintosh Developers"

Sponsored by

Apple **And** MacTech MAGAZINE (For Macintosh Programmers & Developers)

MacWorld Expo / San Francisco

JANUARY 6 - 9, 1998

Moscone Expo Center

AT Developer Central, YOU CAN...

- Talk directly to representatives from Apple and third-party vendors
- Demo developer products and tools
- Get the best deals on tools, training, and developer resource materials

So, no matter if you're developing a departmental system, developing plug-ins for the Internet, or planning the next great Mac OS application or multimedia title, you'll find what you need at **Developer Central**.

BOOTH 3080 • NORTH HALL

Exhibitor List

AAA+ Software
Adobe Systems Incorporated
Aladdin Systems, Inc.
Altura Software, Inc.
Apple Computer, Inc.
- Apple Developer Programs
- Java/MRJ
- Open Transport
- Rhapsody Tools
- WebObjects

Bare Bones Software, Inc.
Bowers Development Corporation
Conix Graphics
Developer Depot
dtF Americas, Inc.
Hotline Communications, Ltd.
LightWork Design
MacTech Magazine
MAGMA
Mathemaesthetics, Inc.
Metrowerks
MindVision Software
Qdea
Quadrivio Corporation
Red Planet Software
Scientific Placement, Inc.
SNAP Innovation Software GmbH
Tenon Intersystems

Sponsored by Apple Computer, Inc. and MacTech Magazine, Developer Central is the premier location for developers at Macworld Expo. This pavilion provides you with a one-stop location to see the latest technologies, gain new and updated product information, purchase products, and talk directly with the developers of the tools you use to create your solutions.

Your development mission may be to create the best new game, or the most efficient way to manage a business. You may be developing in C++ or scripting your way to a solution. You might be a Web site manager or a content developer on the internet road to fame.

Wherever you are going as a developer, the path to success leads directly to Developer Central at Macworld Expo/San Francisco '98.

Visit the **Developer Central** web site
day or night for the all the latest news.

http://www.devcentral.com

NEWSBITS

by Jessica Courtney

SpotCheck 1.0

GenieWorks, LLC announced the release of SpotCheck 1.0 for the Macintosh. SpotCheck is a language-based editor that "knows" the Java language. It is designed to help a Java programmer produce correct code without relying on confusing and untimely feedback from a compiler.

Specifically, SpotCheck identifies syntax errors and semantic errors (undefined names, type mismatches, etc.) — those errors normally returned by a compiler. This analysis is performed after each edit, giving the programmer immediate feedback on errors.

SpotCheck provides a host of additional features, including:

- Smart links to name declarations.
- Cross-referenced Java APIs.
- Editing with popup menus.
- Interfaces to helper apps to compile & run.
- Hierarchical project browsing.
- Color-coded syntax.

<http://www.genieworks.com/>

SiteWarrior Wages War On Web Site Complexity

ProVUE Development has released SiteWarrior, a new program that uses a revolutionary approach for authoring and managing medium to large web sites. This approach makes HTML coding faster, more flexible and more productive by using an ultra fast RAM based database to manage the structure of an entire web site. SiteWarrior automates tedious tasks such as building tables, catalogs, tables of contents, navigation banners, links to neighboring pages, and other repetitive elements within a web site. Unlike most web authoring tools, SiteWarrior does not attempt to eliminate the need for HTML with WYSIWYG features. Instead, SiteWarrior embraces HTML and automates HTML production with custom "HTML factories" controlled by user defined templates.

By holding the entire web site in a single powerful database, all structures and links are updated automatically as pages are added, renamed, or deleted. This guarantees that there will be no broken links within a site. The web site author can search, replace, or check spelling in a single page or across the entire site. SiteWarrior also manages all GIF and JPEG images with a built in image database.

SiteWarrior is designed to work smoothly with other web products. Because SiteWarrior uses a non-WYSIWYG, pure text editor, it never modifies existing page tags in any way. SiteWarrior works with any version of HTML and is compatible with JavaScript and all browsers, CGI's, plug-ins and applets, current and future. The finished web site does not require any special server software, and is compatible with servers on any platform. SiteWarrior also includes automatic FTP uploading for working with remote servers.

<http://www.sitewarrior.com>

Objective-Everything Release 5

TipTop Software, Inc. announced Release 5 of the Objective-Everything development system for OPENSTEP (Rhapsody, Windows, and Mach), including Objective-Python, Objective-Tcl, Objective-Perl, and Objective-Browser.

Objective-Everything provides true language independence for the OPENSTEP (Yellow Box) object model. It allows you to develop programs for OPENSTEP without being restricted by the lack of expressiveness, class libraries, or other limitations of a particular programming language. With Objective-Everything you can use the language best suited for the task at hand, or even mix-and-match languages.

Objective-Everything is a development tool for anyone who develops under OPENSTEP, including programmers who just want to program in straight Objective-C. Objective-Everything is ideally suited for rapid application prototyping and development, application scripting, regression testing, exploratory programming, and much more.

Objective-Browser is a graphical tool which allows you to visualize objects and object-relations in a running program. This is a tool for all Apple Macintosh developers getting familiar with the OPENSTEP development environment!

The browser allows you to view live objects within a running application in many object-specific ways. OB allows you to easily investigate the structure and methods of various objects and classes. For example, classes hierarchy, methods, instance variables, and other object information can be viewed and edited during execution.

OB is extensible. You can provide object-specific browser nodes and custom inspectors for any object in the system. OB is dynamically loaded on demand and can be used from any application.

<http://www.tiptop.com>

NetCloak 2.5 Upgrade

Maxum Development Corporation announced the release of NetCloak 2.5, a major revision to the Web server tool for creating dynamic Web pages. NetCloak, the first commercial Web server tool ever available for Macintosh, has been included on the Apple Internet Server Solution for well over a year, and is now included as a standard component of Apple's AppleShare IP 5.0.2 package. The new version adds dozens of new capabilities, improved support for the wide range of servers available for the Mac OS, better 3rd-party tools support, and more.

NetCloak is now included as a standard component of Apple Computer's complete file and Web server solution, AppleShare IP 5.0.2.
<http://www.maxum.com>

DynaMorph 1.5 Application Server

Morph Technologies, Inc. announced that it has released DynaMorph 1.5, the latest version of its flagship server-side scripting language. DynaMorph enables a website or application developer to employ the power of CGI programming directly inside of HTML documents without the need for advanced PERL or C++ programming skills. With the latest release, Morph Technologies is again morphing the future of technology on the Web by making it even easier to build sophisticated websites and web-based applications.

Version 1.5 of DynaMorph adds database connectivity for all ODBC-compatible databases on Windows NT/95 and Mac servers as well as native database support for UNIX-based databases. Also available with the current release are PC- and Mac-Authorize plug-ins which enable real-time, credit card authorization directly from any webserver.

With DynaMorph, it is easier to perform the tasks associated with website creation and maintenance as well as to develop sophisticated web-based applications which previously required an intimate knowledge of CGI programming. Examples include discussion forums, chat servers, on-line scholastic testing systems, interactive training, adaptive information publishing, product registration, and electronic commerce.

DynaMorph is the only true cross-platform server-side scripting engine, with support for Windows NT/95, Mac OS and UNIX. This enables web development companies (or internal IS/Web departments) to perform site or application development work on Macs or PCs, yet deliver their work to clients running websites, intranets and web-based applications on NT or UNIX platforms. A website or application built using DynaMorph can literally be moved from one computer to another, independent of the operating system, and work automatically with no additional effort required.

DynaMorph not only eases the creation of web pages with dynamic content, but also greatly simplifies the maintenance of static pages. DynaMorph accomplishes this by taking a site creation and maintenance perspective rather than simply one of individual page creation and maintenance.

With over 175 commands that can be embedded into HTML, Java, JavaScript, VRML and other documents, DynaMorph provides the expressiveness and flexibility demanded by sophisticated web site development teams. Yet it also includes many pre-built, easy-to-use templates which make it immediately useful to both programmers and non-programmers alike.
<http://www.morphtech.com>

Roaster Release 4 Product Family: All Java — Cross Platform and Extensible

With its first announcement of detailed information regarding the eagerly awaited Roaster Release 4, Roaster Technologies revealed that the new release is written entirely in Java(tm), utilizes the Java(tm) Foundation Classes, and is the basis for its Data and Enterprise Editions. Release 4 is the cornerstone of the extended Roaster family of development software for high-end professional developers working in Java. The environment is completely extensible, enabling the integration of a wide variety of technologies.

The Release 4 project was originally started using Roaster Release 3 on the Macintosh, heavily utilizing the Internet Foundation Classes from Netscape.

Roaster is one of the first Java development environments to be written entirely in Java, based on JDK 1.1 and written utilizing the Java Foundation Classes. This support gives Roaster users a consistent development experience across all major operating system platforms.

Roaster products products are available through the Developer Depot at <http://www.devdepot.com>. Roaster Release 4 is scheduled to ship at the end of Q1, 1998. The public beta release is scheduled for January.
<http://www.roaster.com/>

Nationwide Lasso Training Course Announced

Blue World Communications, Inc. and Chris Moyer Consulting, Inc. announced the availability of Mastering Lasso I, a product proficiency training course for Lasso 2.0, the leading third party solution for web-enabling FileMaker Pro databases. This two day hands-on course allows users to sharpen their skills and provides users of Blue World's Lasso 2.0 recognition for their knowledge and expertise with professional certification for completing Mastering Lasso I. A follow-up course Mastering Lasso II is scheduled for early 1998.

In addition to Lasso product training, users will receive hands-on experience with FileMaker Pro from Claris Corporation, BBEdit from Bare Bones Software and WebSTAR from Starnine Technologies. User will also receive promotional materials, a Mastering Lasso I certificate, future discounts for training courses offered by Chris Moyer Consulting, future consideration for Blue World's Business Partner Program and qualify for free Mac OS Web software worth over $1,000.
<http://www.fmpro.com/lassotraining/>

OpenBase Updates

OpenBase International, Ltd. has made OpenBase Network and OpenBase Lite available for both the PC-Compatibles and PPC versions of Rhapsody.

OpenBase is a high performance SQL engine that handles all of the complexities of data storage and multi-user communication for end-user applications. Aggressive multi-threading, row level locking, text searching, change notification and variable record length technology makes OpenBase a robust database solution.

OpenBase Lite is a FREE run time license to the SQL database. The purpose of the license is to allow software vendors to sell shrink-wrapped database applications without paying royalties for single licenses.

OpenBase Lite provides a zero administration database interface. An EOF adaptor is included. OpenBase Lite does not include development tools and should not be used to host web applications (this would violate the license agreement).

OpenBase Lite for Yellow-Box on NT will be made available as soon as Yellow-box on NT is available. OpenBase Lite will not be available for OpenStep 4.2 systems.
<http://www.openbase.com>

New Scripting Tool Adds Macro Capabilities

Nombas, Inc. announced the release of the ScriptEase:Integration SDK 4.0 developers' kit for applications and embedded systems. The kit enables application developers to enhance, extend and customize their applications using the flexible, easy-to-use, JavaScript compatible, ScriptEase scripting language. The ScriptEase:Integration SDK allows developers to quickly and simply embed a fully functional script language interpreter into any C/C++ application.

Over 90% of all browser client-side dynamic content on the web is already being developed with JavaScript. By merging ScriptEase with JavaScript, Nombas is making JavaScript available for all classes of software and embedded systems, not just for browsers. ScriptEase:Integration SDK leverages the popularity, power and ease-of-use of JavaScript (an international standard under the name "ECMAScript") to enable developers to customize and extend applications far beyond their off-the-shelf capabilities. ScriptEase:Integration SDK provides the full power of an interpreter engine without the time, engineering or hardware resources required to develop a proprietary macro language. ScriptEase:Integration SDK allows for more flexible applications, improved code re-use, easier portability and significantly reduced application development times.

Incorporating the ScriptEase (JavaScript) language into your application is a fast and straight-forward process. Through simple initializing function calls, the ScriptEase engine is directly integrated into your application. The ScriptEase:Integration SDK's 80 API functions allow the application unlimited interaction with the script language and vice versa. The ScriptEase:Integration SDK offers a safe and controlled environment in which to access application functions and data and can be executed from within a single 64k segment.

Platforms and operating systems supported by the ScriptEase:Integration SDK include Windows 3.1, Windows `95/NT/PPC, Linux, Unix, DOS, OS/2, Mac OS, NetWare, and embedded systems. OS/390 and 100% Pure Java upgrades will be available in Q1, 1998.
<http://www.nombas.com>

Natural Intelligence Spins Off DragStrip; Macintosh Desktop Utility

Natural Intelligence, Inc. announced the sale of DragStrip, its award winning desktop utility for the Macintosh. This action reflects Natural Intelligence's focus on the expansion of its core custom software consulting business, and its newly formed Corporate Java Division. DragStrip will be acquired by Christopher Evans, the original creator of the product, who will upgrade, support and distribute the product as a shareware product.

DragStrip 3.0's new features include the ability to have multiple tabbed pages on a strip, improved support for MacOS 8, including contextual menus, new "Strip" styles, easier configuration, support for URLs and more.
<http://www.natural.com>

Metrowerks Announces First Quarter 1998 Revenues of US$6.2 Million

Company Records Profit of US$237,000, or $0.02 per Share, for Quarter

Metrowerks Inc., a provider of software development tools, today announced its results of operations for the first quarter of its fiscal year ending July 31, 1998 (Q1FYJuly98).

Revenues for Q1FYJuly98 totaled US$6.15 million, an increase of 65% over revenues of US$3.77 million for the corresponding Q1FYJuly97. Revenues for Q4FYJuly97 totaled US$6.23 million. As in previous years, revenues for Q1FYJuly98 were approximately the same as revenues for Q4FYJuly 97.

In Q1 FYJuly98, the Company reported a net profit of US$237,000 (US$0.02 per share on a fully diluted basis), compared to a net loss of US$135,000 (US$0.01 per share) for the corresponding Q1FYJuly97, and compared to net income of US$155,000 for the previous quarter, Q4FYJuly97.

The Company completed a financing during the quarter, whereby the Company will issue 1.25 million shares at $10.08 per share. The Company had a cash balance of approximately $15.5 million on October 31, 1997.

The Company has two main revenue streams; the sale of shrink-wrapped software, or product revenues, and product agreement revenues, whereby companies pay Metrowerks to support their platforms. Product revenues for Q1FYJuly98 comprised 74 percent of total revenues, or $4.53 million.

Cost of goods sold for Q1FYJuly98 were 18% of net revenues compared to 17% for Q4FYJuly97. In Q1FYJuly98, R&D expenses totaled US$2.1 million as compared to US$2.3 million in Q4FYJuly97. R&D expenses were down slightly as the Company has ramped up headcount to a point where less third party consultants are needed for development projects. SG&A and technical support expenses totaled US$2.37 million in Q1FYJuly98 and US$2.44 million in Q4FYJuly97. SG&A expenses were down slightly, as the Company increased its focus on operational expenses.
<http://www.metrowerks.com>

MT

TIPS & TIDBITS

By Steve Sisak

Here is a tip about the use of the GetScrap Scrap Manager trap. When your application is starting up, you may want to check the contents of the scrap or transfer it into your private scrap. However, you should be careful not to call GetScrap until after you've made a couple calls to WaitNextEvent.

This delay is required to allow the Process Manager to move the contents of the clipboard from the previously active application into your application's process context. Also, don't forget that TEFromScrap is just a wrapper around GetScrap, and has the same requirement.

Here is some code to illustrate the problem. Don't do this:

```
InitGraf(&qd.thePort);
...
InitDialogs(0);
TEFromScrap();

for (;;)
  WaitNextEvent(...);
```

Instead, do something like this:

```
switch (event.what)
{
  case nullEvent:
  {
    static int    idleCount;
    static Boolean    gotBootScrap;

    if (++idleCount == 3 && !gotBootScrap)
    {
      TEFromScrap();
      gotBootScrap = true;
    }
  }
}
```

You're probably wondering why the Process Manager doesn't just set up the scrap correctly when it launches a new application? Actually it can't because the current application might have data in a private scrap, which won't be moved into the system scrap until that application receives a suspend event. The suspend event won't be received, however, until after the LaunchApplication trap has created the new process and returned back to the caller. The Process Manager has to wait until the original frontmost application has received that suspend event before it can safely move the contents of the system scrap into the launched application's context.

Eric Schlegel
ericsc@apple.com

"Rather than trying to guess how many nullEvents to wait for, it may be more reliable to hold off importing the scrap (and other initialization) until you receive your first Apple event. That first Apple event will be either 'aevt'/'oapp', 'aevt'/' odoc', or 'ascr'/' noop' and will not be sent until the Process Manager has fully suspended the previous frontmost application." —ed sgs

Eric replys:

"That would have the same effect. It might not always be a better choice than just waiting for the null events, though, depending on whether you already have Apple event handlers. Stickies opens its data file immediately without waiting for an oapp event, so for Stickies and similar apps it makes sense to just use the idle count." —eric

Want to share a tip with the community and get paid for it? Send it in to <mailto:tips@mactech.com>

Send us your tips or we'll install EvenBetterBusError on your machine! On the other hand, we might just pay you ***$25*** *for each tip we use, or* ***$50*** *for Tip of the Month. You can take your award in goods, subscriptions or US$. Make sure any code compiles, and send tips (and where to mail your winnings) to our* ***Tips e-mail address*** *at tips@mactech.com. (See page 2 for our other addresses.)*

Have You Made a Visit to the New MacTech® Web Site?

http://www.mactech.com

Here are some highlights:

Macintosh Development News

Keep yourself informed. The site has all the current breaking news in the development community and even archives past news pieces.

Search the Site

Search like never before. Search and find that article, news piece or source code you're looking for via the new Phantom search engine courtesy of Maxum.

Article Archives

And you can have it all. These archives have thousands of pages of content in them from the histories of MacTech, MacTutor, develop, and FrameWorks magazines.

develop

develop. Need we say more. The entire history of develop, Apple's award winning technical journal. That is 29 issues from 1990 – 1997.

by Nicholas C. "nick.c" DeMello <online@mactech.com>

Rhapsody is a stable, modern, fully preempted and memory-protected multitasking environment with built-in capabilities for symmetric multiprocessing. It's better than sliced bread, kooler than James Dean, and hotter than an LA freeway in July. It's also not finished yet. On the other hand, Mac OS 8 was released in July of 1997, and over 1.2 million people bought a copy in the first two weeks it was on the market. Over 2 million copies were sold by week four, and that number is still growing. This month, we're going to take Mac OS 8 for a test drive.

Kicking the Tires

Apple shipped Mac OS 8 in the August '97 developer mailing, so odds are you have had some time to explore it. If you don't subscribe to the mailing, you may have ordered a copy from the Claris website or got an idea of the new interface features from one of the five online reviews of Mac OS 8 listed in the Yahoo index. In any case, you'll want to keep an eye out for updates and patches. Apple has established a Mac OS 8 website, with a page on late breaking news of any "unintended features" that may crop up.

Speed comparisons between Mac OS 8 and 7.6 are online at the MacSpeedZone. There are MacBench and Speedometer results, as well as real world tests including times for file copying and opening Photoshop files. File copying was enhanced by more than 300%, and 12 mb of data can be deleted in one tenth of the time under Mac OS 8. Enhanced performance and new features are nice, but the real test of a product is it's Easter eggs. The best Mac OS 8 egg involves the crayon color picker tool. Take a close look at the crayons, then set your clock ahead to the year 2001 (even Apple's Easter eggs have no fear of the year 2000). Pop open the color picker again, and you'll see that the crayons are worn down (from 3 years of use no doubt :-). For a list of other Easter eggs, installation tips, and links to Mac OS 8 specific shareware check out Duncan Crombie's Mac OS 8 Downloads and Tips page.

Order Mac OS 8 Online
<http://www3.claris.com/macos8/>
Reviews of Mac OS 8 for the User
<http://www.yahoo.com/Computers_and_Internet/Software/Reviews/Titles/Operating_Systems/MacOS_8/>
Late Breaking News about Mac OS 8
<http://macos.apple.com/macos/latebreak/>
Speed Comparison, Mac OS 7.6 vs 8
<http://macspeedzone.netgate.net/Comparison/OSMacBench.html>
Layne Karkruff's Desktop Consoles Collection
<http://www.blueskyheart.com/dc.html>
Mac OS 8 Downloads and Tips, by Duncan Crombie
<http://www.ozemail.com.au/~dcrombie/macos.html>

Under the Hood

Apple's Technote 1102 is an essential read for programmers taking on the new OS. It presents a table of over 100 hyperlinks to detailed descriptions of Mac OS 8 changes. Valuable enhancements to existing technologies are documented, like the new drag flavors for the Drag Manager which allow you to limit a drag into the finder to only the trash, or control the name of text clippings being moved to the desktop. The technote also provides links to SDK's for powerful new technologies like contextual menus and the appearance manager.

Some effort will be involved in adapting your current product to Mac OS 8. With the new platinum look and the Appearance Manager, new issues of human interface design have surfaced. Additions and changes in the Macintosh Human Interface Guidelines have been compiled and posted to Apple's Inside Macintosh website. An unofficial list of applications that are having trouble with Mac OS 8 is posted at MacOS8.com, scanning this list may give you ideas of where your own code will have trouble. To help you fix any problems, Apple has provided a web page of developer tools for Mac OS 8, including a link to the new MacsBug (only version 6.5.4 or greater operates correctly in Mac OS 8).

One of the more promising new technologies in 8 is contextual menus. Apple's Advanced Technologies Group has used this technology to create Apple Data Detectors. The Internet Address detectors allow you to select text in any document and activate contextual menus (control-click). Detectors parse the text data and identifies any eMail addresses, FTP, HTTP, or newsgroup descriptions, then a contextual pop-up menu allows you to launch the appropriate client tool. Check out the data detectors and the ATG website, to get ideas for your own use of the contextual menus technology.

Technote 1102 on Mac OS 8
<http://devworld.apple.com/dev/technotes/tn/tn1102.html>
Mac OS 8 Modified Human Interface Guidelines
<http://gemma.apple.com/dev/techsupport/insidemac/HIGOS8Guide/thig-2.html>
Mac OS 8 Incompatibility List
<http://www.MacOS8.com/incompatibility.shtml>
Developer Page for Mac OS 8
<http://devworld.apple.com/MacOS8/>
Apple Data Detectors
<http://www.atg.apple.com/research/tech/AppleDataDetectors/>
<http://applescript.apple.com/data_detectors/>

Down the Road

The next incremental update of the Mac OS will be version 8.1, expected between December 1997 and January 1998. Rumors are already circulating, and the latest gossip has been collected at the MacOS Rumors website. However, if you're the one guy out there disappointed in Mac OS 8, there is at least one other option. Yves Lempereur has updated his TRS-80 emulator for Mac OS 8. Space invaders anyone?

Mac OS 8.1 "Bride of Buster" Rumors
<http://www.macosrumors.com/macosinfo.shtml>
Yves Lemprereur's TRS-80 Emulator for Mac OS 8
<http://www.pacificnet.net/~skyrider/trs80.html>

SERVICES

Great Products Deserve Great Docs!

Documentation should be an integral part of your product, not an afterthought. Manual Labor specializes in end-user documentation for Macintosh® products, including Balloon Help and Apple Guide, HTML or PDF files, and the ever-popular "dead tree edition." And while we're documenting every nook and cranny of your product, we can simultaneously provide extensive usability feedback, resulting in fewer bugs, a simpler user interface, reduced support costs, and better reviews.

Contact us today to tap the expertise our team has already brought to companies like Bare Bones Software, id Software, ResNova Software, Management Software, Quasar Knowledge Systems, and Scantron Quality Computers. And see why we're the best choice to document *your* next Mac product.

Jerry Kindall
810/445-9477
kindall@manual.com
http://www.manual.com/

MANUAL LABOR

We Wrote the Book!
Macintosh Technical Communication Experts

The Law Office of Bradley M. Sniderman

California Lawyer focusing on Intellectual Property, Corporate, Commercial and Contract law, as well as Wills and Trusts.

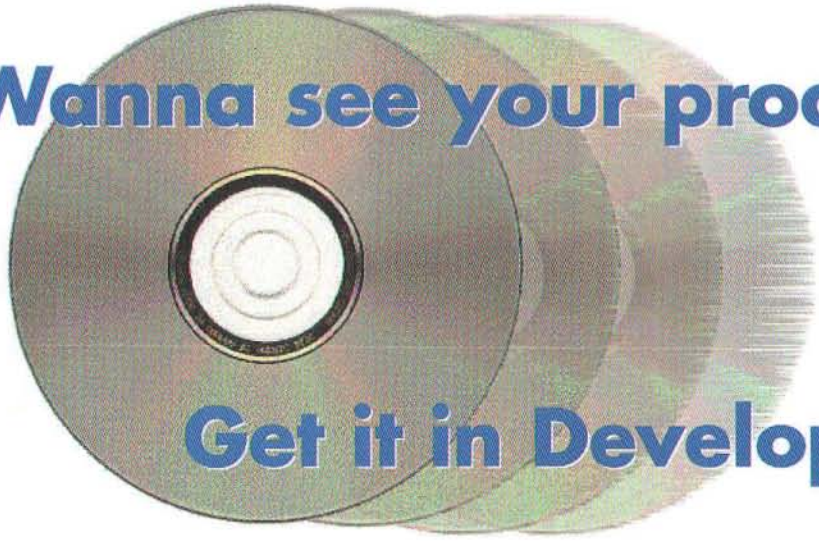
If you are looking to protect your software with Copyright or Trademark protection, or if you need help establishing or maintaining your business, please give me a call or an e-mail. Reasonable fees.

(310) 553-4054
Brad@Sniderman.com

http://www.scientific.com

Professional software developers looking for career opportunities should check out our web site. We offer nationwide employment assistance, resume help, marketability assessment, and never a fee to the applicant. 800-231-5920, Fax 800-757-9003 eMail:das@scientific.com

Scientific Placement, Inc.



Wanna see your product *move?*

Get it in Developer Depot!

The fastest, cost effective way to get product off your shelves.
For information: • Voice: 805/494-9797 • Fax: 805/494-9798 • E-mail: marketing@devdepot.com

ADVERTISER/ PRODUCT LIST

LIST OF ADVERTISERS

LIST OF PRODUCTS

develop

by deeje cooley, San Francisco, CA

Introducing PuppetTime™

Digital actors and how to add new media types to QuickTime

Introduction

3D media is very exciting to use, but for most users is beyond their ability to create. I definitely believe that 3D is the medium of the future. Yet, I am constantly frustrated by the learning curve associated with high-end 3D modeling and animation tools. Sure, for professionals, these tools are the cream of the crop. I want something a little more progressive. I want to manipulate 3D objects that know how to animate themselves and that can interact with each other. Drag a dog onto a stage, then tell it to wag its tail. Drag a cat onto a stage, then tell it to walk around. Put the two together, and tell the dog to chase the cat. In short, I don't want to make 3D objects, I want to use 3D objects to create other things. That's why I've created the PuppetTime architecture.

PuppetTime is an open architecture for digital actors, built on top of QuickTime. What, you may ask, is a digital actor? You may have heard the term before. In its most basic definition, a digital actor is a graphic representation on the computer screen that can accept messages to animate itself. I use the term puppet to mean digital actor, because I want to emphasize the metaphor of virtual strings controlling a shape's appearance and implied behavior.

Here's an example: on my computer screen I have a humanoid-shaped puppet and a list of commands. When I click on a command, it is sent to the puppet, which then responds with some kind of activity. If I click on "walk" and then click on a new location on the screen, the puppet will "walk" to that new location. If I click "wave", the puppet will wave to say hello. Different puppets might animate themselves in different ways to represent "walk" or "wave"; the power lies in the fact that "walk" and "wave" are now abstracted out and any number of puppets can understand these commands while presenting a unique visual appearance for each.

There are many companies now developing digital actors for use in movies and games, but as of yet there is no proposed standard framework that might make the commercial acceptance and distribution of digital actors feasible. The PuppetTime architecture attempts to address this need.

PuppetTime uses the Component Manager and QTAtoms, defines a new component interface called puppets, and includes both a derived media handler and a movie import component. The PuppetTime framework is designed with a philosophy similar to QuickTime: it contains a set of toolbox routines for manipulating the PuppetTime media data, as well as a number of extensible components and component interfaces. Much like Sprites and QuickTime Music Architecture in QuickTime, PuppetTime can be used by itself in your applications, and it can also be contained in QuickTime movies alongside other media types like music, text, sound, and video.

You are highly encouraged to have a copy of the PuppetTime Sample Code on hand while reading this document, as I'll refer to its contents often. You can obtain the sample code at <http://www.puppettime.com/>. A royalty-free license is available for the PuppetTime runtime, and third-party and co-development is highly encouraged.

Basic PuppetTime Architecture

The PuppetTime architecture is implemented using several QTML technologies and defines three key elements: puppets, events, and the conductor.

deeje cooley is passionate about music, visualization, and cinema. PuppetTime is the result of this passion, which he's been developing independently for almost two years now. In this spare time, he answers QuickTime questions in DTS at Apple Computer, Inc. You can reach him at deeje@pobox.com.

Puppets

Puppet components are defined and implemented using the Component Manager 3.0 and display themselves using QuickDraw 3D 1.5.x. The puppet component interface and several of the built-in puppets are discussed in detail below.

Events

A PuppetTime event is a QTAtom data structure that contains information about a command or action that a puppet should perform. When a puppet receives an event, it pulls out the relevant information and (often) performs some form of animation. A stream of events can come from numerous sources, such as from a network connection, or from a QuickTime movie track. The standard PuppetTime event format, toolbox routines, and some basic event vocabularies are discussed below.

Conductor

The PuppetTime conductor component acts as the glue between events and puppets. At its basic level, the conductor creates the QuickDraw 3D environment, instantiates a number of puppets into the environment, and then receives a stream of events from an external source and re-directs them to the individual puppets. Again, the PuppetTime conductor is discussed below.

Goals of PuppetTime

There are several key design goals for PuppetTime to ensure its acceptability and future growth.

Open Architecture

PuppetTime is designed to be an open architecture. The format for PuppetTime events structures allows for a variable number of bits of information, so that new events can be defined and existing events augmented. The puppet component interface allows new puppets to be added seamlessly to existing PuppetTime-savvy applications.

Compact Data Format

The PuppetTime event format is crafted to allow for the widest range of event vocabularies possible, while keeping an eye towards compactness. PuppetTime uses events as meta-data to recreate a scene at runtime, and as such, events are much smaller than pre-rendered, compressed image samples.

Internet-ready

PuppetTime is designed with web- and internet-savvy applications in mind. For example, on-line 3D comic strips would be possible by downloading a set of puppets once, then delivering new episodes on web pages as QuickTime movies. Each episode movie can be significantly smaller than a pre-rendered 3D scene, because it only contains a sequence of events. The puppets themselves can be designed and implemented such that they can update themselves with new capabilities on the fly in numerous ways.

Scalable Performance

PuppetTime performance scales with improvements in CPU speed and internet bandwidth. Because puppets animate themselves in real-time, their visual displays can improve with faster computer systems. Also, faster Internet connections allow for richer, higher-fidelity event streams.

QuickTime Integration

PuppetTime is designed to work as a new media type within QuickTime. To start, PuppetTime includes a MIDI file importer and a media handler, which allows PuppetTime event streams to be stored and played back within a QuickTime movie, alongside other media types (e.g. music and text).

Technology Map

Figure 1 shows the basic QuickTime architecture with the integrated PuppetTime media type and its related components.

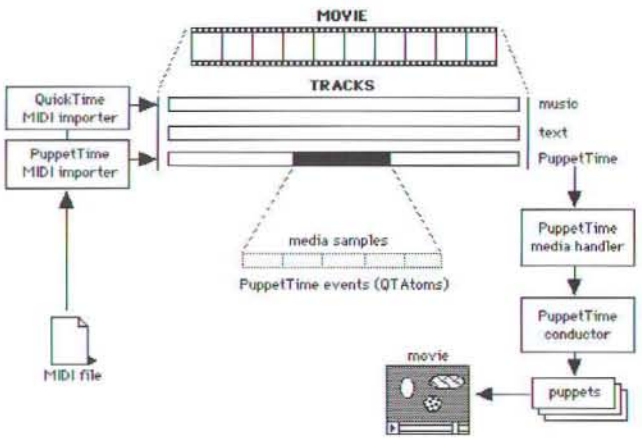


Figure 1. *PuppetTime inside QuickTime.*

Events

Let's begin our detailed discussion of PuppetTime with events. As mentioned above, a PuppetTime event is a QTAtom structure which contains bits of information that describes an action or command to a puppet.

QTAtoms

QTAtoms are structures that store a variable number of name-data pairs. They are similar in concept to AppleEvent records, except that QTAtoms do not store data type information, and the API is available for all platforms that QuickTime supports. The QuickTime 3.0 developers guide describes QTAtoms in detail, and can be found at <http://quicktime.apple.com/>. As shown in **Figure 2**, QTAtoms can be nested inside one another to create hierarchical data structures.

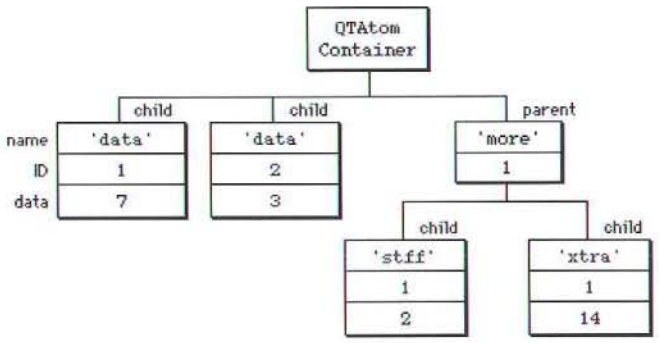


Figure 2. *A typical QTAtom structure.*

Listing 1 shows how to create the QTAtom structure shown in **Figure 2**. Note that proper error checking is not included in the listings below, but you should always check for programmatic and runtime errors in your code.

Listing 1: Building a typical QTAtom structure

```
QTAtomContainer   aContainer = nil;
QTAtom            anAtom = nil;
long              aLong = 0;

   // create the QTAtomContainer
anError = QTNewAtomContainer(&aContainer);

   // add some name-data pairs to the root
aLong = 7;
anError = QTInsertChild(
      aContainer,          // the container
      0,                   // the atom, zero = root
      'data',              // the name
      1,                   // the ID
      0,                   // the index of name-ID pairs
      sizeof(aLong),       // the size of the data
      &aLong,              // the pointer to the data
      nil);                // returns a ref to the new QTAtom

aLong = 3;
anError = QTInsertChild(aContainer, 0, 'data',
            2, 0, sizeof(aLong), &aLong, nil);

   // create an empty atom
anError = QTInsertChild(aContainer, 0, 'more',
            1, 0, 0, nil, &anAtom);

   // add some atoms to it
aLong = 2;
anError = QTInsertChild(aContainer, anAtom, 'stff',
            1, 0, sizeof(aLong), &aLong, nil);

aLong = 14;
anError = QTInsertChild(aContainer, anAtom, 'xtra',
            1, 0, sizeof(aLong), &aLong, nil);

   // make sure to dispose of it when you're done
anError = QTDisposeAtomContainer(aContainer);
```

Basic Structure

Thus, a PuppetTime event is a QTAtom structure with a well-defined set of name-data hierarchy, shown in **Figure 3** (QTAtom IDs are not used by the PuppetTime toolbox routines, and will be omitted from the following figures).

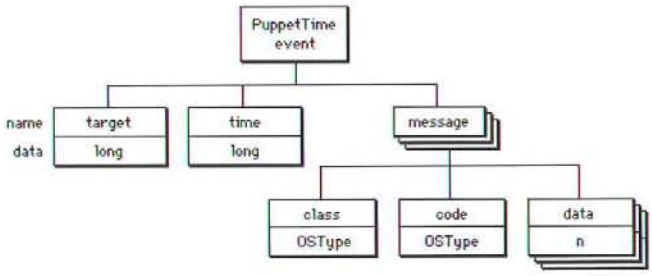


Figure 3. *The PuppetTime event structure.*

A PuppetTime event includes the following atoms:

- target: this atom contains an ID of the target puppet for this event. Puppet IDs are assigned to puppets at runtime, so any puppet can be used, and the event always goes to the puppet with the given ID.
- time: this atom contains the time that this event should occur at. This time value will be either relative to the start time of a movie or to the system clock. The puppet is responsible for queuing this event until the given time has arrived. A queuing method is provided to any puppet that wants it (explained later).
- messages: each event contains one or more messages. Each message contains a message class/code combination and zero or more parameters. In this way, a number of messages can be sent to a puppet with only one event.
- class: contains the message class being invoked (e.g. 'core' or 'musi').
- code: contains the message code, (e.g. 'walk' or 'wave').
- data: each message can contain a number of parameters, as defined by the creator of the event suite. The parameters can augment the resulting behavior and/or animation (e.g. speed of walk, or exaggeration of wave).

There are constants defined in the header file PuppetTimeEvents.h for the event and message names used to build a PuppetTime event, to ensure that all events have the same structure.

PuppetTime Toolbox Routines

As noted above, you can use QuickTime toolbox routines to build QTAtoms as PuppetTime events, as long as you structure the QTAtoms in the basic format described. Because the format is so specific, there are a number of PuppetTime toolbox routines that make creating and parsing PuppetTime events easy. Look at the file PuppetTimeEvents.h for a complete list of available APIs.

Music Events

As an example, let's describe a class of events that represent music. **Figure 4** shows a typical music event structure.

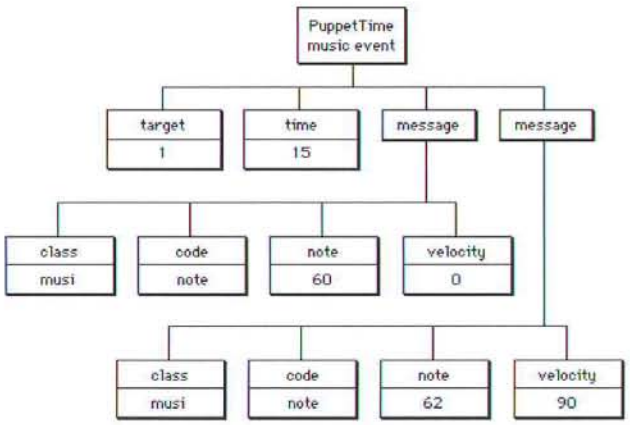


Figure 4. *A typical music event.*

Listing 2 shows how to use the PuppetTime toolbox routines to build the event shown in **Figure 4**. Notice the PuppetTime toolbox streamlines the hierarchical nature of the event for you.

Listing 2: Creating a music event

```
QTAtomContainer MakeAMusicEvent()
{
  SInt32          instrument = 1;
  SInt32          eventTime = 15;
  UInt16          noteNumber;
  UInt16          noteVelocity;
  QTAtomContainer anEvent = nil;
  QTAtomContainer aMessage = nil;
  OSStatus        anError = noErr;

    // create a new message
  aMessage = PTNewMessage(kPTInstrumentClass, kPTNoteEvent);

    // insert the parameters
  noteNumber = 60;
  noteVelocity = 0
  anError = PTSetProperty(aMessage, kNoteNumber,
                sizeof(noteNumber), &noteNumber);
  anError = PTSetProperty(aMessage, kNoteVelocity,
                sizeof(noteVelocity), &noteVelocity);

    // create the event
  anEvent = PTNewEvent(instrument,
              eventTime,
              aMessage);

    // add the second message
  noteNumber = 62;
  noteVelocity = 90;
  anError = PTSetProperty(aMessage, kNoteNumber,
                sizeof(noteNumber), &noteNumber);
  anError = PTSetProperty(aMessage, kNoteVelocity,
                sizeof(noteVelocity), &noteVelocity);

  anError = PTSetNthMessage(anEvent, 0, aMessage);
  anError = PTReleaseMessage(aMessage);

    // do something with the event, like add it to a track

  anError = PTReleaseEvent(anEvent);

  return anEvent;
}
```

Optimizations

To minimize the size of PuppetTime event streams, there are a number of optimizations that can be made when storing or transmitting events.

The first optimization is the concept of default values. When an event is defined by an author in a header file, certain parameters will be defined to have default values. When a developer creates an event, she can omit certain parameters to save space. When the recipient of an event goes to read those parameters and finds none, she can assume a default value. So for the music event example above, the default value for the velocity is zero, and that "note off" messages can contain one less parameter. The savings can add up quickly in a PuppetTime track that represents a song visually. Default value optimizations should be done by the creator of the events.

The second optimization is the concept of event flattening. Often times an event will contain only one message, so the contents of the message atom (class, code, and parameters) are moved into the event atom, and the message atom is removed. The PuppetTime toolbox routine PTOptimizeEventList performs event flattening.

Puppets

Now we turn our attention to PuppetTime puppets. PuppetTime defines a puppet component interface in the file PuppetTimeComponents.h, and also includes a number of puppet components that are used throughout the PuppetTime environment.

The Puppet Component Interface

Listing 3 shows the component interface for puppet components.

Listing 3: The puppet component interface

```
pascal ComponentResult PuppetInitialize
                (PuppetComponent puppet,
                 ConductorComponent aConductor);

pascal ComponentResult PuppetSetTimeFormat
                (PuppetComponent puppet,
                 UInt32 eventTimeFormat);

pascal ComponentResult PuppetIdle
                (PuppetComponent puppet,
                 UInt32 atMediaTime);

      // message routines
pascal ComponentResult PuppetProcessActionEvent
                (PuppetComponent puppet,
                 QTAtomContainer anEvent);

pascal ComponentResult PuppetProcessMessage
                (PuppetComponent puppet,
                 UInt32 atMediaTime,
                 QTAtomContainer aMessage);

      // QD3D routines
pascal ComponentResult PuppetSubmit
                (PuppetComponent puppet,
                 TQ3ViewObject theView);

pascal ComponentResult PuppetGetGroupObject
                (PuppetComponent puppet,
                TQ3GroupObject* aGroup);

pascal ComponentResult PuppetGetTranslateObject
                (PuppetComponent puppet,
                TQ3TransformObject* aTransform);

pascal ComponentResult PuppetGetCameraObject
                (PuppetComponent puppet,
                 Rect* graphicsBox,
                 TQ3CameraObject* aCamera);
```

The key routines are described below.

- PuppetInitialize: this routine is called when an instance of the puppet is created. The puppet should initialize its internal structures, which usually includes creating some geometries in QuickDraw 3D that serve as the puppet's basic visual representation.
- PuppetProcessActionEvent: this routine is called when an event is dispatched to a puppet. The puppet should check the time of the event and queue the event if the current time is less than the event time.
- PuppetProcessMessage: when a puppet decides to execute an event, it should send all messages to this routine. This abstraction between processing an event and a message will become clear below when we talk about the base puppet component.
- PuppetIdle: this routine is called repeatedly while the puppet is active, and the puppet should do whatever processing is necessary. Often times, in response to an event, the puppet will animate itself, and this is the routine that should handle the next "frame" of the animation sequence. It is up to the puppet to decide how to implement it's animation. Most puppets perform their animations by adding, changing, and removing geometries in the QD3D environment.
- PuppetSubmit: this routine is called for each puppet when the 3D environment is being drawn. The puppet is responsible for submitting its QuickDraw 3D geometries.

Base Puppet

As you can see, there are a number of routines in a puppet component, and every puppet should implement all of them. But most puppets need the same internal organizations, like a queue for events that aren't quite ready for processing. Also, processing events and pulling out messages is the same for most puppets.

To make the process of creating a new puppet for PuppetTime easier, most developers can create a derived puppet component. A derived puppet uses the services of a base puppet component as a delegate to its own code. Similar in concept to a base media handler or a base image decompressor, the base puppet component implements the basics of a puppet, and leaves the specifics of the geometries and animations to the developer.

To create a derived puppet component, a developer must implement the following puppet component routines: PuppetOpen, PuppetClose, PuppetInitialize, PuppetIdle, and PuppetProcessMessage.

The PuppetOpen routine should make an instance of the base puppet component and set derived puppet component to be the target. Listing 4 shows a simple derived PuppetOpen routine.

Listing 4: A derived PuppetOpen routine

```
pascal ComponentResult
  PTBlockyPuppetOpen(ComponentInstance self)
{
  ComponentResult     result = noErr;
  PTBLPrivateGlobals**storage = NULL;

  storage = (PTBLPrivateGlobals**)
        NewHandleClear(sizeof(PTBLPrivateGlobals));
  if (storage != NULL)
  {
    // store our globals in the component instance
    SetComponentInstanceStorage(self, (Handle) storage);
    (**storage).self = self;
    // get the Blocky media handler component
    (**storage).delegate = OpenDefaultComponent(
                              PuppetComponentType,
                              BasePuppetComponentType);
    ComponentSetTarget((**storage).delegate, self);

    // initially we target ourselves
    (**storage).target = self;
  }

  return (result);
}
```

In PuppetClose, make sure to release your instance of the base puppet.

In PuppetInitialize, your puppet component must call through to the base component, and then create some geometries. Listing 5 shows a simple derived PuppetInitialize routine. Notice that the base puppet creates the QD3D group object, and that the derived puppet asks for it using the puppet component routine PuppetGetGroupObject.

Listing 5: A derived PuppetInitialize routine

```
pascal ComponentResult
PTBlockyPuppetInitialize(PTBLPrivateGlobals** storage,
                         ComponentInstance aConductor)
{
  TQ3GroupObject    aGroup = nil;
  TQ3GeometryObject myBox;
  TQ3BoxData          myBoxData;
  TQ3SetObject        faces[6];
  short               face;
  TQ3ColorRGB       faceColor;
  TQ3ColorRGB       faceSeeThru;
  ComponentResult   anError;

  anError = PuppetInitialize((**storage).delegate,
                             aConductor);
  anError = PuppetGetGroupObject(
              (**storage).target,
              &aGroup);

  // set up the colored faces for the box data
  myBoxData.faceAttributeSet = faces;
  myBoxData.boxAttributeSet = nil;
  // set up some color information
  faceColor.r = faceColor.g = faceColor.b = 0.8;
  faceSeeThru.r = kNoteTransparency;
  faceSeeThru.g = kNoteTransparency;
  faceSeeThru.b = kNoteTransparency;
  for (face = 0; face < 6; face++)
  {
    myBoxData.faceAttributeSet[face]
                = Q3AttributeSet_New();
    ::Q3AttributeSet_Add(myBoxData.faceAttributeSet[face],
                kQ3AttributeTypeDiffuseColor,
                &faceColor);
    ::Q3AttributeSet_Add(myBoxData.faceAttributeSet[face],
                kQ3AttributeTypeTransparencyColor,
                &faceSeeThru);
  }
  // set up te basic properties of the box
  ::Q3Point3D_Set(&myBoxData.origin,
                  0, -(6 * kNoteSize), 0);
  ::Q3Vector3D_Set(&myBoxData.orientation,
                  0, 12 * kNoteSize, 0);
  ::Q3Vector3D_Set(&myBoxData.majorAxis,
                  0, 0, kNoteLength);
  ::Q3Vector3D_Set(&myBoxData.minorAxis,
                  kNoteWidth, 0, 0);

  // create the box itself
  myBox = ::Q3Box_New(&myBoxData);
  ::Q3Group_AddObject(aGroup, myBox);
  ::Q3Object_Dispose(myBox);
```

```
    // dispose of the objects we created here
  for( face = 0; face < 6; face++)
  {
    if (myBoxData.faceAttributeSet[face] != nil)
      ::Q3Object_Dispose(myBoxData.faceAttributeSet[face]);
  }

  return anError;
}
```

In the routine PuppetIdle, you can do whatever idle time processing you like. Just make sure to give the base puppet some idle time as well. Listing 6 shows how.

Listing 6: A derived PuppetIdle routine

```
pascal ComponentResult PTBlockyPuppetIdle(PTBLPrivateGlobals**
storage,
                        UInt32 atMediaTime)
{
  ComponentResult   anError;
  anError = PuppetIdle((**storage).delegate,
                       atMediaTime);
  for (short i = 0; i < kNumberOfNotes; i++)
  {
    if ((**storage).fNotes[i] != nil)
    {
      anError = (**storage).fNotes[i]->Idle(atMediaTime);
    }
  }
  return anError;
}
```

When the base puppet decides to pull an event from its queue, it reads each of the messages inside it and sends them to PuppetProcessMessage. This is where your puppet can receive its messages and perform its animations. Notice that the switch statement defaults to calling back into the base puppet, which can handle certain basic messages on its own (e.g. "locate at"). Listing 7 shows an example.

Listing 7: A derived PuppetProcessMessage routine

```
pascal ComponentResult
  PTBlockyPuppetProcessMessage(
              PTBLPrivateGlobals** storage,
              UInt32 atMediaTime,
              QTAtomContainer aMessage)
{
  ComponentResult anError = noErr;
  OSType          messageCode;

  ::PTGetMessageCode(aMessage, &messageCode);
  switch (messageCode)
  {
    case kPTNoteEvent:
    {
      anError = ProcessNoteMessage(storage, aMessage);
      break;
    }

    default:
      anError = PuppetProcessMessage(
                  (**storage).delegate,
                  atMediaTime,
                  aMessage);
      break;
  }

  return anError;
}
```

As you can see, a base puppet handles a lot of the details for you, allowing you to concentrate on implementing your puppets visual appearance and animations. Also, note the object-oriented nature (i.e. inheritance and overriding) of using a base puppet inside your puppet. The Component manager was designed specifically for this kind of use.

Camera Puppet

Another important puppet in PuppetTime is the camera puppet. This is a derived puppet which provides a view of the PuppetTime world to the user. Any puppet can have a camera view associated with it, although it's not required. The camera puppet is special in that it has no geometry associated with it, and simply provides a view.

The file PuppetTimeCameraEvents.h defines a vocabulary for camera control, and the file PuppetTimeEvents.h includes the core vocabulary for basic movement. This means that the view can be changed by sending "move" events to the camera puppet. Just like all other PuppetTime events, these "move" events can be generated at runtime based on user input devices (e.g. a joystick) or can be stored along with other events in an event stream (e.g. panning during playback of a movie). At this time, there is only one camera puppet allowed; future versions of PuppetTime will expand the role and use of camera-enabled puppets.

Creating your own puppets

There are several puppets with sample code available in the SDK, which demonstrate the proper way to use the base puppet component. Use these example projects as the basis for your puppet development.

Make sure that you edit the 'thng' resource, and choose mixed- or upper-case constants for the subtype and manufacturer fields. At this time, there are no flags defined for puppets, so zero them out for now.

When implementing puppets, you'll have to decide what vocabularies to support. There are several sets of vocabularies already defined in PuppetTime, such as core and music. The base puppet handles a number of the core events for you.

You're also free to create your own vocabularies, but you should use your manufacturer code as the message class; you'll also have to generate PuppetTime tracks using your vocabularies.

Music Puppets

For music puppets, the file PuppetTimeMusicEvents.h contains all the constants for the music vocabulary. The file PuppetTimeMusicEvents.c includes several utility routines to easily build music events.

The PuppetTime MIDI import component, discussed below, creates PuppetTime tracks using the music vocabulary. This allows a user to quickly generate PuppetTime content by simply importing MIDI files into QuickTime movies using applications like MoviePlayer.

Conductor

Next we examine the heart of PuppetTime, the conductor. It is the central object that binds the puppets to the drawing environment and to the incoming event stream.

3D Environment

When an instance of the conductor component is created, it in turn instantiates a number of QuickDraw 3D objects to set up a drawing environment, including a renderer, viewer, context, etc.

Each puppet is responsible for its own geometries, yet this information needs to be communicated to the conductor at some point. This is done when the conductor is instructed to draw: each puppet gets a chance to submit its geometries (and other objects) to the QuickDraw 3D rendering loop maintained by the conductor.

Event Dispatching

Besides being responsible for the overall display, the conductor is also responsible for dispatching events from an incoming events stream to the puppet instances.

There is a class of events specific to the conductor, like the 'cast' event. Events targeted to the conductor have a target ID of zero. A cast event has the structure shown in **Figure 5**.

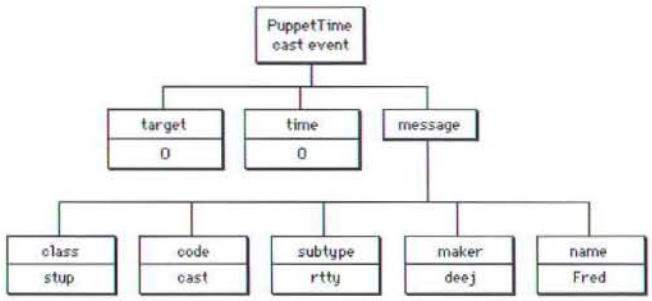


Figure 5. *A cast event structure.*

The cast message contains the following fields:

- kPuppetSubType: this field contains the subtype of the puppet component to instantiate, or cast. This field corresponds to the subtype field in the 'thng' resource that defines your puppets. When casting a puppet, the conductor will search for a component where the type is 'PTpt' and the subtype is the value in this field.
- kPuppetMaker: this optional field allows you to further identify your puppet component, which matches the manufacturer field in your 'thng' resource. When casting your puppets, you should use this field to avoid name collisions, which can result in the wrong puppet being cast.
- kPuppetName: this field is also optional, but if it's present, it will be used in future version of PuppetTime for things like onscreen identification and event targeting via name.

When the conductor receives 'cast' events, it examines the contents of the event to determine which puppet to instantiate, then creates and stores a puppet instance in its internal array.

Cast events are among the first events passed to a conductor. Without them, there would be no puppets visible and nothing to dispatch further events to. You can use the routine PTAddCastingEventToList to easily add a cast event to an event list. Note that a cast event doesn't tell the puppet where it should be when it is created; therefore, you should also add a locate event to the event stream using the routine PTAddLocateEventToList.

We'll talk about cast events as they pertain to QuickTime movies below.

Current Camera

The conductor has the concept of a current camera, which is itself a puppet. When a conductor first initializes, and after it has created the QuickDraw 3D drawing environment, it casts a camera puppet and assigns it a special ID kPTDefaultCamera. This gives the conductor an initial view in which to draw.

Events can be targeted to the default camera by using an ID of kPTDefaultCamera Because the camera object uses the base puppet, it understands many of the core vocabulary, like "move to" and "turn". The first version of PuppetTime supports only one camera, but future versions will expand on this.

QuickTime Integration

The first part of this article described how PuppetTime is structured around QTAtoms and puppet components. Now lets focus on how PuppetTime is integrated with QuickTime. The initial release of PuppetTime allows for basic creation and playback of QuickTime movies containing PuppetTime tracks.

PuppetTime Tracks

A PuppetTime track in a QuickTime movie has the type 'PTmh'. Each sample in a PuppetTime track is a QTAtomContainer structure containing a list of PuppetTime events. The events in this list represents at minimum 2-3 seconds worth of animation; the duration of each sample can be much larger. This assures that disk access is minimized for better playback performance.

Playing PuppetTime Tracks

In QuickTime, each track type has a corresponding media handler component which handles the playback of the track contents. So, a video track is managed by an instance of the video media handler, and a sound track is managed by an instance of the sound media handler.

The PuppetTime media type is no different: a PuppetTime track is managed by the PuppetTime media handler, which is a derived media handler. When QuickTime opens a movie and finds a PuppetTime track, it searches for the corresponding PuppetTime media handler (by finding a component of type 'mhlr' and a subtype equal to the track type, in this case 'PTmh') and creates an instance.

Being a derived media handler, many of the functions are handled by the base media handler component. The PuppetTime media handler does handle certain routines itself, and the most notable are MediaInitialize and MediaIdle.

MediaInitialize function

During movie initialization, QuickTime calls the media handler routine MediaInitialize. Here, the PuppetTime media handler creates an instance of the PuppetTime conductor and tells it about the visual dimensions of the track.

Next, it looks for some cast data associated with the track. The cast data is stored separately so that the cast of the movie can be easily changed. For example, in a PuppetTime track that contains music events, the actual puppets used during playback can be changed by modifying the cast data. The resulting visual representation will be different while the underlying event stream remains valid. Use the routines PTGetCast and PTSetCast to get and set the cast data for the track.

MediaIdle function

If the conductor is the heart of PuppetTime, then the MediaIdle function is the heartbeat of a PuppetTime track. This routine is responsible for reading in samples from the underlying media and passing them off to the conductor for dispatch. It also gives idle time to the conductor so that it can draw.

Creating PuppetTime Tracks

Of course, playing a PuppetTime track is only useful if you have a PuppetTime track. Creating a PuppetTime track in a QuickTime movie is simple. As explained above, each sample is a list of events; in this way the samples are spaced apart such that the disk isn't accessed too often. Listing 8 shows the sample description record, which is rather uncomplicated. Listing 9 demonstrates the concepts behind adding a PuppetTime media to a movie.

Listing 8: PuppetTime sample description

```
typedef struct PTMHDescription {
  long size;            // Total size of struct
  long type;            // kPTMediaType
  long resvd1;
  short  resvd2;
  short  dataRefIndex;
  long version;
} PTMHDescription, *PTMHDescriptionPtr,
**PTMHDescriptionHandle;
```

Listing 9: creating a PuppetTime track

```
Track                   myTrack;
Media                   myMedia;
QTAtomContainer         anEventList;
PTMHDescriptionHandle   aDesc = nil;
TimeValue               sampleTime;

myTrack = NewMovieTrack(theMovie,
            (long) myWidth << 16,
            (long) myHeight << 16,
            0);

  // create the media for the track
myMedia = NewTrackMedia(myTrack,  // the track
            kPTMediaType,         // the type of media
            aTimeScale,           // time scale
            nil,                  // data ref
            (OSType) nil);        // type of data ref

anEventList = PTNewEventList();
anError = PTAddLocateEventToList(anEventList,
                      1,    // target
                      0,    // time
                      0,    // x
                      0,    // y
                      0);   // z
anError = PTAddNoteEventToList(anEventList,
                      1,    // target
                      5,    // time
                      60,   // note
                      95,   // velocity
                      0);   // duration (0=forever)
anError = PTAddNoteEventToList(anEventList,
                      1,    // target
                      65,   // time
                      60,   // note
                      0,    // velocity (0=off)
                      0);   // duration (0=forever)

aDesc = (PTMHDescriptionHandle)
          NewHandleClear(sizeof(PTMHDescription));
(**aDesc).size = sizeof(PTMHDescription);
(**aDesc).type = kPTMediaType;
(**aDesc).version = kPTMediaVersion;

  // Start editing session
anError = BeginMediaEdits(theMedia);

  // add the data to the media
anError = AddMediaSample(theMedia,
              (Handle) anEventList,  // the sample
              0L,                    // offset
              GetHandleSize((Handle) anEventList),
              65,                    // duration of sample
              (SampleDescriptionHandle) aDesc,
              1,                     // number of samples
              0,                     // sample flags
            &sampleTime);            // returned time

  // end editing session
anError = EndMediaEdits(theMedia);

  // append to the track
anError = InsertMediaIntoTrack(
                theTrack,     // the track
                -1,           // where to insert
                0,            // where in the media
                65,           // how much media to insert
                1L << 16);    // the media rate
```

Of course, there are routines in the PuppetTime toolbox that make creating PuppetTime tracks even easier, like PTAddPuppetTimeSample and PTSetEventListForTrack.

PuppetTime movie import component

Users should have an easy way to create PuppetTime tracks from abundant existing content. The initial release of PuppetTime focuses on music visualization, and includes a movie import component that converts MIDI files into PuppetTime tracks.

QuickTime already includes a movie import component for MIDI files. The trick is to hook into the QuickTime import component such that while it is creating a music track, PuppetTime gets a chance to create a PuppetTime track alongside it.

This can be done by capturing the MIDI import component and replacing it with the PuppetTime import component. This is a step beyond just delegating to a component. Capturing means that the PuppetTime import component gets exclusive use of the MIDI import component, and takes the latter out of the Component Manger's current registry.

Also, PuppetTime wants to capture the MIDI import component at startup time, so that whenever QuickTime starts to import a MIDI file, and regardless of which application is calling QuickTime, the PuppetTime MIDI import component gets to do its magic.

Capturing a component at runtime takes a bit of finesse. First, the 'thng' resource must be properly configured: the type and subtype of the PuppetTime import component must match the component being capturing (in this case 'eat ' and 'Midi'). Next, the cmpWantsRegisterMessage flag is set to true, which tells

the Component manager that the PuppetTime import component wants its Register routine called at startup. The rest of the component flags should be the same as the component being capturing. Finally, the PuppetTime movie import component is a PPC native component, so the componentHasMultiplePlatforms flag is set to true. This tells the component manager to find the component in the extended 'thng' structure.

Now that the component successfully captures the MIDI import component, it needs to override the MovieExchangeImportFile function. This routine calls through to the captured and delegated MIDI import component, which proceeds to create the music track from the MIDI file. After that routine returns, the PuppetTime import component then re-reads the MIDI file and creates a PuppetTime track, converting MIDI data structures into PuppetTime events using the music vocabulary. The code could have just as easily read the newly created music track. Either way, without any extra effort on the user's part, a new movie is created that contains both a music track and a PuppetTime track.

To make the user experience complete, the PuppetTime import component also overrides the MovieExchangeDoUserDialog routine. In this case, however, it doesn't call through to the MIDI import component, but puts up its own Options dialog instead.

Sample Code

In the PuppetTime Sample Code I've included slightly altered versions of the PuppetTime media handler and the PuppetTime movie import component for your review. You'll note that I've changed all occurrences of the subtype and manufacturer fields to 'XXXX' and 'YYYY'. If you choose to use these samples for the basis of your own projects, please change the constants to something more suitable. Also, please don't re-use my constants for the various PuppetTime components, particularly 'PTmh' for my media handler and media type; this will allow me to continue developing PuppetTime without external complications.

Future Direction

Much like the QuickTime architecture, PuppetTime is designed with future growth squarely in mind.

More QuickTime integration

With the initial release of the PuppetTime engine, only two QuickTime-related components are included: a media handler and a movie import component. As PuppetTime continues to develop and mature, more QuickTime components will be added.

- Sequence grabber channel: A PuppetTime sequence grabber channel will work with the sequence grabber component to capture PuppetTime tracks in real-time from a number of input devices, like keyboards and joysticks. It could also be used to capture a network event stream, such as in a multi-user game environment.
- Movie Controller: PuppetTime is well integrated with other QuickTime media types, but the existing user experience doesn't allow the user/viewer to move around and among the puppets currently being displayed. A PuppetTime movie controller will add a 'trackpad' like control alongside the other QuickTime movie controller controls that allows the user to move the camera puppet while the movie is playing.
- Movie Info Panel: Not necessarily a formal part of the QuickTime framework, a PuppetTime movie info panel will allow users to change the puppets in the cast for a PuppetTime track.

Cross-platform

Of course, creating a new media type, especially for web and internet applications, isn't as compelling unless it works on Mac and Windows. Near-term future development will focus on bringing the core toolbox and component functionality to both platforms under QuickTime 3.0.

Consumer Applications

And, of course, the PuppetTime architecture exists so that developers can create applications that create and edit the PuppetTime media type. A couple of consumer-level applications that I'd like to see happen are the Puppet Builder and the Puppet Scene Maker.

The Puppet Builder application would allow a user to create new puppets, giving them shapes and simple animations, and matching animations to events.

The Puppet Scene Maker would allow a user to create a scene with dialog in 3D. Drag puppets from a cast window onto a stage window, then enter dialog in the script window. Drag actions from a vocabulary window onto the script window to add movement, nuances, etc.

Bibliography and References

- Wang, John. "Somewhere in QuickTime: Derived Media Handlers" *develop,* The Apple Technical Journal, issue 14 (June 1993), pp. 87-92.
- Guschwan, Bill. "Somewhere in QuickTime: Dynamic Customization of Components" *develop,* The Apple Technical Journal, issue 15 (September 1993), pp.84-88.
- *Inside Macintosh: QuickTime,* by Apple Computer, Inc. (Addison-Wesley, 1993).
- *Inside Macintosh: QuickTime Components,* by Apple Computer, Inc. (Addison-Wesley, 1993).
- *3D Graphics Programming With QuickDraw 3D,* by Apple Computer, Inc. (Addison-Wesley, 1995).

URLs

The QuickTime homepage is at <http://quicktime.apple.com/> and the QuickTime developer homepage is at <http://quicktime.apple.com/dev/>.

The PuppetTime homepage is at <http://www.puppettime.com/>, where you can download the latest docs, runtime, and SDK.

Acknowledgments

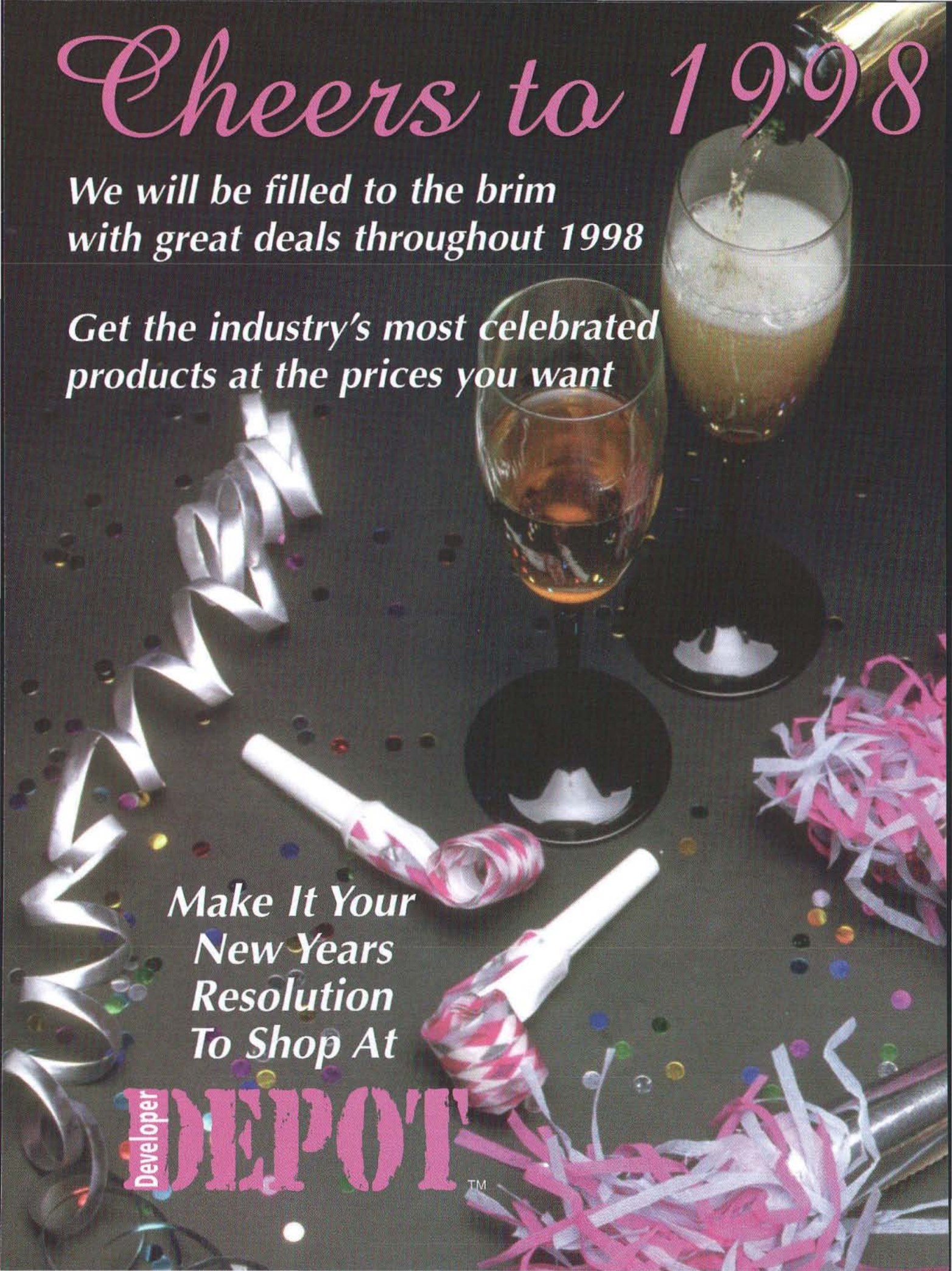
Thanks to Joel Cannon, Scott Kuechle, Gregg Williams, Kathryn Donahue, Steve Cooley, Tony Gentile, and Jason Downs.

Developer DEPOT™

Developer Depot PO Box 5200 Westlake Village CA 91359-5200
800/MACDEV-1 • Outside the U.S. & Canada: 805/494-9797 • Fax: 805/494-9798
E-mail: orders@devdepot.com • Web Site: http://www.devdepot.com

- ✓ World renowned customer service
- ✓ Hundreds of developer products
- ✓ Order by phone, E-mail, fax or through our continually updated Web site
- ✓ Satisfaction and lowest price guaranteed

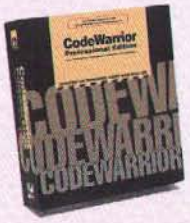
INCLUDES EVEN MORE APPLE BRANDED DEVELOPMENT PRODUCTS!


Cheers to 1998
We will be filled to the brim
with great deals throughout 1998
Get the industry's most celebrated
products at the prices you want
Make It Your
New Years
Resolution
To Shop At
Developer
DEPOT
TM

ANNOUNCEMENT

Developer DEPOT™

We Are Starting Off
The New Year Right
Where We Want To Be



MACWORLD EXPO

San Francisco

January 6-9, 1998

Come visit us at

The Moscone Convention Center
in The Developer Central Pavilion
located in the North Hall

TABLE OF CONTENTS

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

One & 2-Day SHIPPING (call for details)

Order Toll-free
800-MACDEV-1
(800 622 3381)

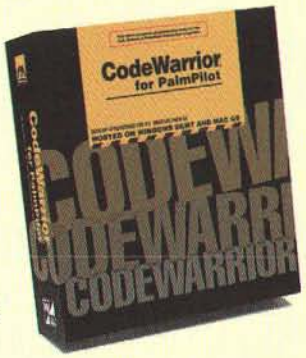
Developer Depot 30 day Money Back, Price and Satisfaction Guarantee

Developer Depot products are sold with a 30 Day money back guarantee on user satisfaction, lower prices and against defects. If, for any reason, you are not satisfied or find the same product at a lower price within 30 days, please call Customer Service at 800-MACDEV-1 and request a Return Merchandise Authorization (RMA) number to get a full refund or the difference in price (where applicable). You must return undamaged product at your expense, including all its original packaging, documentation and the blank warranty card if applicable. Developer Depot will replace defective product upon receipt of the defective merchandise. Please remember to back up your data before installation of any new hardware, software, or peripherals; we cannot be responsible for any lost data. Policies, item availability, and prices are subject to change without notice. The price in effect when we receive your order will be the price that you are charged. We are not responsible for any typographical errors in this or any other catalog, nor for any misstatements from any vendor. Purchase orders are not accepted without prior approval. Call for more information.

metrowerks

CodeWarrior for PalmPilot

by Metrowerks

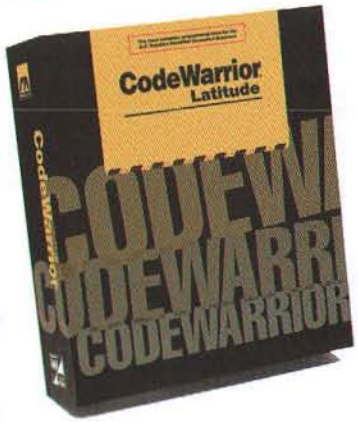


CodeWarrior for PalmPilot is the first complete set of tools which enables you to develop for the U.S. Robotics PalmPilot connected organizer, from the comfort of your PC or Macintosh computer. All the tools you need are included: the award-winning CodeWarrior IDE, compiler, linker, source-level debugger, GUI builder, and other tools, plus online documentation and reference materials. Includes one free product update and free technical support with registration.
(SCWPALM) Our Price **$369**

NEW PRODUCT!

CodeWarrior Latitude

by Metrowerks

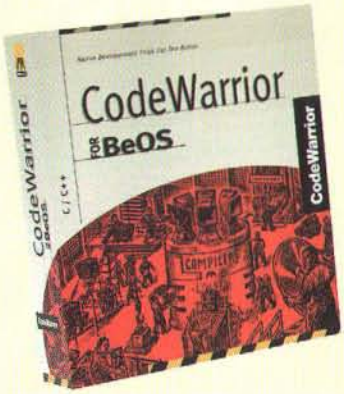


Don't throw away the investment you have made in your Mac OS application! With the new DR2 release of CodeWarrior Latitude, you can now port your Mac OS application to Rhapsody, as well as Silicon Graphics IRIX and Sun Solaris. At the heart of Latitude is a set of shared libraries which performs the functions of the Macintosh API. You recompile your Mac OS source code, linking it with the Latitude libraries to produce a native application. As the Rhapsody API evolves, so will Latitude. Registered users of CodeWarrior Latitude will receive all developer releases, the first full release, plus one additional product update, to ensure that you have access to the most up-to-date Rhapsody porting tools available.
(SCWLAT) Our Price **$399**

Be OS™

CodeWarrior for BeOS 3

by Metrowerks

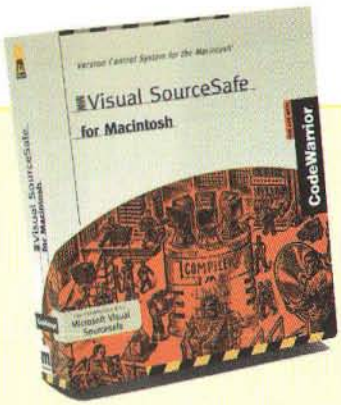


- Start programming for the new, innovative Be Operating System (BeOS) with complete set of Codewarrior tools
- BeOS-native Integrated Development Environment (IDE) with all the familiar CodeWarrior features at your fingertips
- A BeOS PowerPC compiler and linker, an editor w/syntax color and styling, and a source-level debugger
- BeOS header and libraries, complete documentation, useful C++ classes, and sample code
- Now includes Java support
- The BeOS Preview Release allows you to run and program for the BeOS on a 603 or 604 PCI based PowerMac

(SCWFB) Our Price **$299**

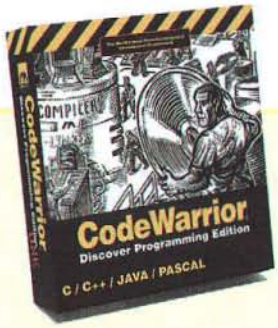
MW Visual SourceSafe Release 5

by Metrowerks



- Source code control system, plug-in to the CodeWarrior IDE or stand-alone Client application
- Compatible with Microsoft Visual SourceSafe version 5.0
- Cross-platform support (Mac, Windows, UNIX)
- Configuration management in excess of 4 billion files
- Over 8,000 files and sub-projects in a single sub-project
- Registered users receive one year of free technical support and one free update

(SMWVSS) Our Price **$499**



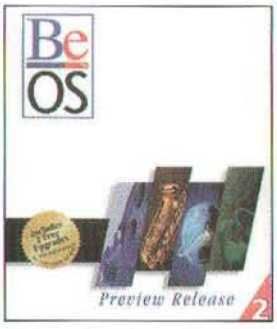
CodeWarrior Discover Programming Edition

by Metrowerks

- Learn to program in C/C++/Java/Pascal
- Full-featured programming product–not a "lite" edition
- Online books and tutorials
- Hosted on either Mac OS or Windows 95/NT

CodeWarrior Discover Programming Edition offers an unbeatable combination of full-featured programming tools, online books and instructional materials, all at a great price. Whether you want to learn the basics of programming or add a new language to your skill set, Discover Programming puts the information and tools you need at your fingertips: the award-winning, easy-to-use CodeWarrior Integrated Development Environment (IDE), four of the most popular programming languages, online books, online tutorials, sample source code and free technical support (with your registration). Discover what you can learn with CodeWarrior, buy your copy today!

(SDPED) Our Price **$79**

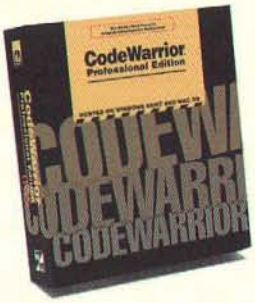


BeOS Preview Release 2

by Be, Inc.

As the Internet spreads and electronic media becomes more prevalent, the high-performance needs of digital content design and the complex, aging architectures of current mainstream operating systems are coming into conflict. The BeOS is the first operating system designed to unlock the door to much more powerful personal computers, and extract more performance from the systems we use today.

(SBEOS) Our Price **$49**



NEW PRODUCT!

CodeWarrior Professional Release 2

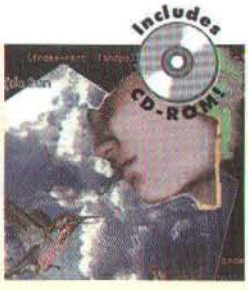
by Metrowerks

- Includes Windows 95/NT and Mac OS versions of the CodeWarrior IDE
- Supports C, C++, Java and Pascal
- Develop for Windows 95/NT on x86 and Mac OS on 68K/PowerPC
- V2 Project Manager supports multiple open projects, subprojects, multiple targets per project, and threaded execution
- Support for JDK 1.1.3 in CodeWarrior Java

More platforms, more languages, more options: CodeWarrior Professional is designed to give you the tools you need for serious, industrial-strength programming. CodeWarrior Professional is the only Integrated Development Environment (IDE) in which you can edit, compile and debug C, C++, Java and Pascal programs for multiple target processors and operating systems. CodeWarrior's compilers produce fast, highly optimized code for Windows 95/NT running on x86, or Mac OS running on 68K or PowerPC processors. CodeWarrior Professional features the CodeWarrior IDE Version 2. The revamped Project Manager supports multiple projects open simultaneously, multiple targets per project, and threaded execution, and it's significantly faster. CodeWarrior Professional also includes online books, documentation, and reference materials, as well as tutorials and sample code. We support your development efforts with one free update and free world-class technical support for a year with registration.

(SCWPRO) Our Price **$449**

Order Toll-free
800-MACDEV-1
(800-622-3381)



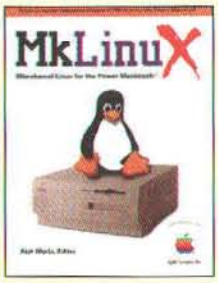
Macintosh Common Lisp 4.0

by Digitool, Inc.

Macintosh Common Lisp provides users with a rich set of object-oriented dynamic language features making it especially well-suited for rapid prototyping, custom development for business and education, scientific and engineering applications, and academic research.

- Power PC native environment & compiler, full Macintosh support
- CLOS, the standard Common Lisp object system
- Interactive dynamic environment, multiple processes
- Automatic memory management and self-typing data
- Ephemeral garbage collector, smaller application footprint
- Compiles with Common Lisp industry standard and smart programmable tools, 110+ mb of user contributed code
- Complete on-line documentation (manual sold separately)
- Software license and registration card

(SMCLISP) Our Price **$675**



MkLinux: Microkernel Linux for the Power Macintosh

by Prime Time Freeware

MkLinux is a native port of Mach 3 and the Linux 2.0 kernel, complemented by hundreds of commands from BSD, GNU, and X11. It runs on most (NuBus and PCI bus) Power Macintosh systems; Performa, PowerBook, and multiprocessor ports are currently under development. MkLinux is robust, powerful, freely distributable, and source code compatible with most other Linux systems. It provides a full suite of development tools, support for AppleTalk, HFS, and Objective-C, and access to a vast amount of free software. MkLinux is a great way to "come up to speed" on Mach, UNIX, and Rhapsody.

- MkLinux user community supports FTP and web servers, development and porting efforts, and several mailing lists
- The Apple sponsored reference release contains a wealth of introductory and reference material on Linux, Mach, NeXT, and the Power Macintosh
- Includes free 3.0 upgrade

(BMKLINUX) Our Price **$49**

VIP–BASIC: Visual Interactive Programming in BASIC

by Mainstay

Now you can create full-featured, stand-alone Macintosh and Power Macintosh applications in standard BASIC code! VIP-BASIC 2.0 is the fastest way to program your Macintosh.

- Rapid application development environment with application framework, mix and match: VIP-BASIC high-level subprograms
- Import pre-existing BASIC code: automatically integrate BASIC code, export C Code for compiling: automatically convert your BASIC code to C for compilation with Metrowerks' CodeWarrior

(SVIPBASIC) Our Price **$195**

VIP–C: Visual Interactive Programming in C

by Mainstay

Now you can create full-featured, stand-alone Macintosh and Power Macintosh applications in just minutes. VIP-C 2.0 is the first rapid application development system for creating complete Macintosh programs in standard ANSI C.

- Includes powerful, tightly integrated visual debugger, Import pre-existing C code: automatically integrate C code with a current project
- Includes full-featured mini database: (ltd to 32K) of the powerful VIP-BASIC database manager gives you everything you need to setup royalty-free, multi-user database applications

(SVIPC) Our Price **$295**

Pro Fortran

by Absoft Corporation

Absoft Pro Fortran combines native F90, VAX compatible F77, and C/C++ compilers into a single, easy to use environment. All compilers are link compatible and operate through a common interface.

- Graphical debugger, browsers, array display, performance profiler, linker, MRWE application mainframe
- MIG graphics library, Absoft Create Make, several utilities, the latest version of MPW and illustrated documentation
- Whole array operations, modules, interface blocks, and user-defined types or data structures
- Dynamic memory allocation and new control constructs
- F90 is link compatible with Absoft F77, C++, MrC and CodeWarrior
- It is fully compatible with Toolbox, MPW tools, and most third-party products

(SAPROF) Our Price **$899**

SmalltalkAgents for Macintosh

by Quasar Knowledge Systems, Inc.

- An Integrated Development Environment (IDE) based on QKS Smalltalk.
- "Live" direct manipulation of your objects
- Dynamic, interactive and iterative development process
- Easy and full access to the features of the Mac OS(tm) and Mac Toolbox
- Link your non-Smalltalk code fragments with Code Fragment Manager (CFM) support
- Cross-reference, access, view, and manipulate your code and objects with a sophisticated database for source code management
- Includes an Application Delivery Toolkit(tm) (ADT) that allows you to create royalty-free, standalone, double-clickable applications

(SSTA) Our Price **$395**

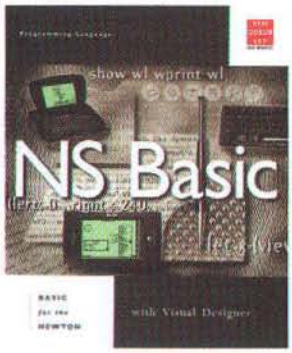
ObjectMaster Professional Edition

by Altura Software, Inc.

Object Master is an innovative programming environment that provides all the necessary tools to write, organize, and navigate through source code.

- Write code using the most robust source code editor available on the desktop
- Organize source code into projects to quickly access and manipulate all files
- Navigate through source code using intuitive graphical Browser windows

(SOMPE) Our Price **$399**

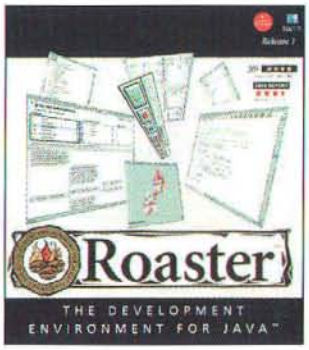


NS BASIC 3.6 for the Newton with Visual Designer

by NS BASIC Corporation

- A fully interactive implementation of BASIC programming language
- Runs entirely on the Newton – no host is required
- Create files, access the built in soups, and the serial port for input and output
- Work directly on the Newton, or through a connected Mac/PC and keyboard
- Get the **BASIC Internet Tool**, available at no charge to NS BASIC users from www.nsbasic.com
- Release Notes with sample code are available from the same location
- Runs on any Newton MessagePad 130 with NS BASIC and the Newton Internet Enabler. Also runs on MP 1201s with NOS 2.0 that have full memory available
- Write short programs to access News, mail and the web

(SNSBASIC) Our Price **$99**



Roaster

by Roaster Technologies, Inc.

Get the most out of Sun's Java™ programming language with this powerful development environment.

- Features include: ability to build stand-alone Macintosh applications or applets; visual interface builder; ability to create cross-platform zip files; powerful Java debugger; wizard for quickly creating Java applets or applications; JDBC included; Java object database included; ability to call AppleScripts from Java; Just-in-time (JIT) compiler; JDK 1.0.2 support (JDK 1.1 support with a free web-based update); class tree and hierarchical class browser; much more!
- Software includes: Over 300 example applets and applications; Netscape Internet Foundation Classes; Object Design's PSE for Java; OpenLink Software's JDBC Drivers; OpenSpace Java Generic Library; Microline Component Toolkit Lite 3.0; much more!
- Requirements: Runs on 68k or PowerPC, CD-ROM
- Price includes all web-based updates. Release 3 owners now have access a web-based update to Release 3.1 with JDK 1.1 support.

(SROAST3) Our Price **$99**

Also see Internet Related. page 13

TENON INTERSYSTEMS

CodeBuilder

by Tenon Intersystems

CodeBuilder is a powerful and unique Macintosh software development tool for porting existing apps or developing new, advanced applications on Power Macs and Power Mac clones.

- A powerful Macintosh software development tool suite of C, C++, Objective-C, Java, Ada, and Fortran development tools
- Complete UNIX & X development environment for developing UNIX or Macintosh apps
- Includes compilers and source-code debugger for Objective C, and C, C++, Ada 95 and Fortran 77
- Web & internet scripting tools: Perl, MacPerl, tcl/tk, bash, sh, and csh
- Supports Rhapsody kernel APIs and Rhapsody TCP sockets

(SMIOCODEB) Our Price **$149**

WAIT... There's More!

Here's a list of all available products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	OUR PRICE
LPA MacProlog Developers Edition	SLPAD	995.00
LPA MacProlog Programmers Edition	SLPAP	495.00
LS Fortran Pro	SLSFORT	595.00
LS Fortran Plug-In	SLSFPI	199.00
Mac FORTRAN II	SFORT2	595.00
Power MachTen–UNIX	SM10PPC	695.00
Presenting Magic Cap	BPRESMAGIC	15.25
Think Pascal 4.0	SPASCAL	165.00

PowerKey Pro Model 200
by Sophisticated Circuits

PowerKey Pro Model 200 lets you start up and shut down your Mac and up to five peripherals with a single keystroke. Two groups of switched outlets let you control some peripherals separately. PowerKey also features phone ring startup which lets you access your Mac while on the road. Powerful scheduling features let you control your outlets with "hot keys" or perform tasks unattended. Start up your computer at any time of the day or night, open applications and run AppleScripts or QuicKeys. Add the optional Server Restart Option and you can even restart crashed servers automatically! System Requirements: Mac with ADB port, System 7 or later. Telephone features require analog phone line.

(HPKEY2) Our Price **$99**

PowerKey Pro Model 600
by Sophisticated Circuits

PowerKey Pro Model 600 is "the world's smartest power strip!" Start up and shut down your Mac and peripherals with a single keystroke. Includes six individually-switched outlets, with manual switches and indicator lights. Powerful scheduling features let you control outlets with "hot keys" or perform tasks unattended. Start up your computer at any time of the day or night, open applications and run AppleScripts or QuicKeys. Complete telephone controllability lets you start up the computer, switch outlets or run complex events using custom touch-tone commands. For a limited time, Model 600 includes the Server Restart Option. Restart crashed servers automatically! System Requirements: Mac with ADB port, System 7 or later. Telephone features require analog phone line.

(HPKEY6) Our Price **$199**

ObjectSet Mail SDK
by Smartcode Software

- Powerful C++ classes for integrating Inter e-mail in your applications
- Helps you write software that can share m with other leading e-mail products
- Royalty-free MIME, SMTP, and POP3 APIs Macintosh, Windows, and Unix
- Gives you the most robust MIME parser ar encoder available
- Ideal for use in Internet and Intranet environments
- Comes complete with samples with documented, reusable source code
- Free standard technical support

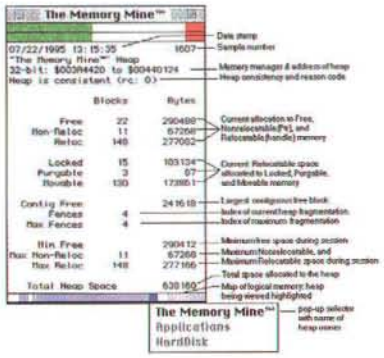
(SOSMSDK) Our Price **$495**

NetMinder Ethernet
by Neon Software

NetMinder Ethernet is a software-only protocol analyzer which captures and decodes a full range of Ethernet protocols including IP, AppleTalk, NetWare, NetBIOS and DECnet. Features include:

- Sophisticated long-term monitoring with HTML output
- Intuitive and powerful filtering capabilities
- Automatic mapping of names to Ethernet, AppleTalk, and IP Addresses
- Rules-based engine for detecting unusual network conditions
- Customizable graphs for bandwidth utilization and packet rates

(SNETMD) Our Price **$715**



Memory Mine
by Adianta, Inc.

- Monitor heaps, identify problems such as memory leaks, and stress test applications
- Active status of memory in a heap is sampled on the fly: allocation in non-relocatable (Ptr), relocatable (Handle) and free space is shown, as are heap corruption, fragmentation, and more
- Allocate, Purge, Compact, and Zap memory lets users stress test all or part of a program

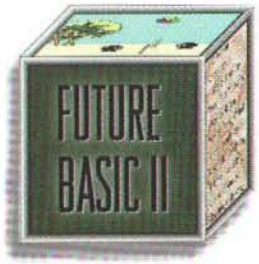
(SMEMMINE) Our Price **$99**

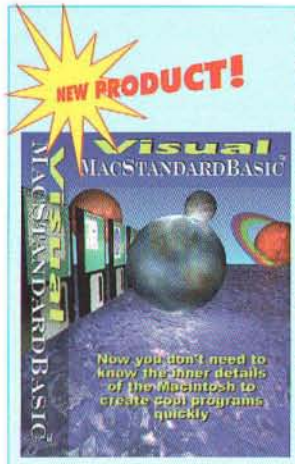
Future BASIC II
by Staz Software

FutureBASIC II is the award winning leader in Macintosh BASIC programming.

- Source level debugger and Interactive compiler/editor
- Multi-file Project manager and Multi-file find and replace
- Super fast compilation, 32 bit clean, and System 7.x savvy
- QuickBASIC converter
- Getting Started manual with over 500 example files
- Full support of standard BASIC

(SFBASIC2) Our Price **$229**





Visual MacStandardBasic 3.0

by ZCurve Software

Visual MacStandardBasic is the new standard for creating both 68K and Power Macintosh applications.

- Applications can be visually created in minutes
- Visual controls such as command buttons, text boxes, list boxes, radio buttons, check boxes, scrollbars, icons, pictures and timers can be created and modified instantly
- Use color graphics, animations, movies, sounds and speech in your programs
- Console text window option helps converting older BASIC source code from other platforms
- Online tutorial, manuals, sample projects get you programming quickly

(SVMACSB) Our Price **$99**

Spotlight

by Onyx Technology, Inc.

Spotlight is a stand alone debugging aid that performs memory protection (arrays, heap accesses, outside your heap, low mem, etc), discipline checking on toolbox calls, and leaks detection.

- Spotlight is sold on an annual subscription basis
- The subscription service provides all updates
- Includes maintenance releases for one year after the initial purchase or renewal date.

(SSPTLT) Our Price **$199**

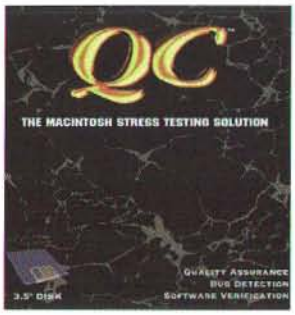
QC

by Onyx Technology, Inc.

High performance runtime stress testing for applications.

- Tests include heap checks, purges, scrambles, handle/pointer validation, dispose/release checks, write to zero, de-reference zero as well as other tests like free memory invalidation and block bounds checking
- Extremely user friendly – ideal for non-programmer testers
- Also available in Japanese

(SQC) Our Price **$99**



MacA&D 6.0

by Excel Software

- Structured analysis and design
- Object-oriented analysis and design
- Real-time and multi-task design
- Data and screen modeling
- Integrated code editing and browsing
- Multi-user dictionary and requirements
- Code to design diagrams for C, C++,etc.
- Design diagrams to code for C, C++, etc.
- State modeling diagrams and tables
- Use cases with traceability

(SMACADP) Our Price **$1995**

Celestin

Apprentice 6

by Celestin Company

Apprentice 6 is a high-quality CD-ROM collection of over 600 megabytes of up-to-date source code, utilities, and info for Mac programmers. All of the source code and utilities are completely new or updated for this release.

- Frontier 4.1, the highly-acclaimed scripting environment
- More PowerPlant AND many more PowerPC samples
- Cool new languages and environments added (Clean, Eiffel, F, Tcl-Tk)
- Hot new demos from leading Mac development companies

(SAPPRENT6) Our Price **$35**

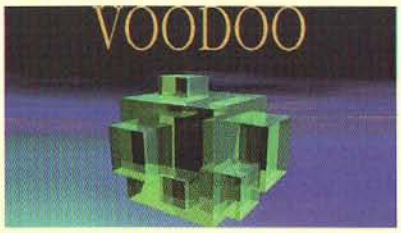
StoneTable 68K/PPC

by StoneTablet Publishing

StoneTable is a powerful and professional replacement for the List Manager used by developers worldwide. Version 3.0 is a new release with many improvements including better clipboard and drag/drop integration with other applications.

- Available for use with CodeWarrior C & Pascal
- Includes libraries for 68K (A4 & A5) and PowerPC
- An LTable-like class is provided to incorporate StoneTable into the PowerPlant environment

(SSTONEFAT) Our Price **$199**



VOODOO 1.8

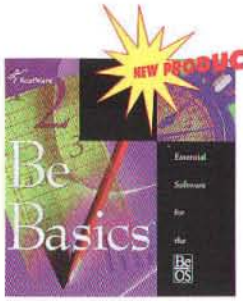
by UNI SOFTWARE PLUS

- Stand-alone version control tool for all sorts of projects (software development, documentation, design, CAD, publishing, etc.)
- Smooth integration with Metrowerks CodeWarrior and BBEdit.
- Simple and clear management of variants and revisions of entire projects (not only of single files)
- Easy-to-use graphical project browser gives access to all versions that were ever stored.
- Recording of the complete history (who made which changes when and why)
- View differences between versions (not only for text files!)
- Efficient delta storage of arbitrary files (text as well as non-text files) gains savings of 95 % and more
- Administration of users with hierarchical access rights
- Configurable local file locking (Finder flag or 'ckid' resource)
- Scriptable, essential parts PowerPC native

Single license (SVOODO01) **$229**
2 pack (SVOODO02) **$359**
5 pack (SVOODO05) **$799**
10 pack (SVOODO10) **$1369**
20 pack (SVOODO20) **$2399**

Additional pricing available on request.

SEE RELATED CATEGORY: Dev. Environments



Be Basics
by BeatWare

Be Basics is the first productivity suite to combine the essential tools you use everyday with the incredible power of the BeOS. Be Basics offers the BeOS developer the ability to:

- Combine text, tables, graphs and pictures to create compelling documents
- Work with multiple files simultaneously without degrading performance
- View all layout, style and content changes as they happen
- Import Microsoft Excel files
- Plug-in third party graphs and filters

Be Basics requires the BeOS, 16 MB of RAM, 2 MB available hard disk space and a 256-color display adapter.
Price includes all major and minor upgrades through version 2.0 via electronic distribution.
(SBEBASIC) Our Price **$69**

CodeBuilder
by Tenon Intersystems

TENON INTERSYSTEMS

CodeBuilder is a powerful and unique Macintosh software development tool for porting existing apps or developing new, advanced applications on Power Macs and Power Mac clones.

- A powerful Macintosh software development tool suite of C, C++, Objective-C, Java, Ada, and Fortran development tools.
- Complete UNIX & X development environment for developing UNIX or Macintosh apps
- Includes compilers and source-code debugger for Objective C, and C, C++, Ada 95 and Fortran 77
- Web & internet scripting tools: Perl, MacPerl, tcl/tk, bash, sh, and csh
- Supports Rhapsody kernel APIs and Rhapsody TCP sockets

(SMIOCODEB) Our Price **$149**

B-Tree HELPER 2.2
by Magreeable Software

- Inexpensive database engine for Macintosh programmers in C source code
- Uses contiguous fixed length blocks
- Expands the file as necessary and contracts files when possible
- Inserts and deletes keys in one or more B-Trees
- Finds keys equal to, less than, or greater than a given value in a few hundredths of a second
- Finds lists of records whose keys are equal to, less than, or greater than a given value or are in a range of values

(SBTREE) Our Price **$149**

BBEdit 4.5
by Bare Bones Software

BBEdit 4.5 is a powerful, easy-to-learn text and HTML editor that offers developers and HTML authors the ability to build on its core functionality to suite their specific needs through its plug-in architecture and scripting capabilites. This new version includes: a visual table tool that speeds page and site development, contextual menu support for Mac OS 8, improved storage for 'grep' patterns, scriptable HTML authoring preferences and more. It still provides: unparalleled searching muscle with support for both 'grep' style and advanced literal searches, the ability to quickly compare differences between files or entire folders, integrated support for Symantec's IDE, Metrowerks CodeWarrior, THINK Reference 2.x, MPW Toolserver and most other environments and a heck of a lot more.
(SBBEDIT) Our Price **$119**
Also see Internet Related. page 12

Guide Composer™ 1.2
by StepUp Software

STEPUP SOFTWARE

- Create powerful Apple Guide help systems for any new or existing Macintosh application
- Provides a WYSIWYG development environment: Guide content is developed in Guide windows
- Design topics, phrases, and panels in the same format as the user will use them
- Features are WYSIWYG interface, Topics, phrases, and hierarchical phrases, Coach marks, Fully-Integrated with Apple's Guide Maker (distributed with Guide Composer), compiles scripts automatically, PICTs in Panels, Generated Guide scripts are modifiable
- FREE Update to all registered Guide Composer users. Demo is available at http://www.guideworks.com/

(SGCOMP) Our Price **$99**

SEE RELATED PRODUCTS: AppleGuide Complete, Danny Goodman's AppleGuide Starter Kit, Real World AppleGuide

Step-Up Installer Pack
by StepUp Software

- Package of several Installer "atoms" that let developers incorporate graphics, sounds, file compression and custom folder icons into installation scripts
- Compression formats supported are Compact Pro & Diamond
- Each atom also available separately
- Compression requires additional licensing

(SINSTALL) Our Price **$219**

ScriptGen Pro
by StepUp Software

- Installer script generator which requires no programming or knowledge of Rez
- Supports StepUp's InstallerPack, StuffIt decompression, Compact Pro decompression, custom packages, splash screens, network installs, and resource installation

(SSCRPTGEN) Our Price **$169**

BeSpecific 3

(Third in a series)
by Adamation

NEW PRODUCT!

The best-selling BeSpecific CDROM series provides the best in BeOS shareware across a wide range of application areas, including:

- Productivity programs
- Programming tools
- Commercial demos
- Latest source code
- Graphics Games

Newsgroup archives (comp.sys.be) Developer mailing list (Be DevTalk) BeSpecific 3 is brimming with useful BeOS Preview Release-compatible software. A "must have" companion for all users of the BeOS.

(SBESPEC) Our Price **$39**

Pilot Attaché Disk 1

(First in a regular series)
by Adamation

NEW PRODUCT!

The Pilot Attaché CDROM, designed for the popular US Robotics Palm Pilot organizer provides you the best in Pilot shareware. Extend your Pilot's capabilities across a wide range of application areas, including:

- Personal productivity tools
- Programming tools
- Utilities
- Newsgroup information
- Games

Fully tested, Pilot Attaché's shareware treasure throve will help you get the most out of your Palm Pilot. When you travel with your Pilot, don't forget your Attaché. Pilot Attaché...your passport to success.

(SPATCHE) Our Price **$29**

Water's Edge Software

Tools Plus libraries + framework

by Water's Edge Software

Easily create compact, fast running, professional looking applications and plug-ins*. Tools Plus lets you create virtually any user interface element with a single routine, and it transparently provides a robust infrastructure to make all your pieces work together as an application.

- Simplifies programming and thins source code
- Automates all standard GUI elements
- Thousands of extras, from floating palettes and tool bars to powerful picture buttons
- Includes numerous 3D grayscale options
- Over 1/2 MB of custom fonts, icons, cursors, and other resources
- Includes SuperCDEFs world-class controls (an $89 value) free

(STOOLCW) Our Price **$249**
CodeWarrior Gold
(C/C++ & Pascal, 68K & PPC)

(STOOLCWB) Our Price **$199**
CodeWarrior Bronze (C/C++ & Pascal, 68K)

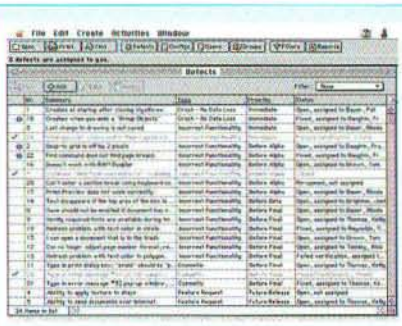
(STOOLSYMT) Our Price **$199**
Symantec (THINK) C/C++ and THINK Pascal (68K)

(STOOLSYM) Our Price **$149**
Symantec (THINK) C/C++ (68K)

(STOOLPAS) Our Price **$149**
THINK Pascal (68K)

*CodeWarrior required to write plug-ins

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.



TestTrack-Bug Tracking the Macintosh Way

by Seapine Software, Inc.

- Tracks bugs, feature requests, test configurations, users, and more
- Includes notifications, security, a powerful filter mechanism, and multiple reports
- Links your testers, engineers, documentations staff, and project managers together to ensure all bugs are identified, fixed, and documented
- Eliminates the need to build custom bug tracking solutions using general purpose database tools
- Supports single- and multi-user bug databases (additional licenses required to use multi-user features)

(STETR) Our Price **$129**

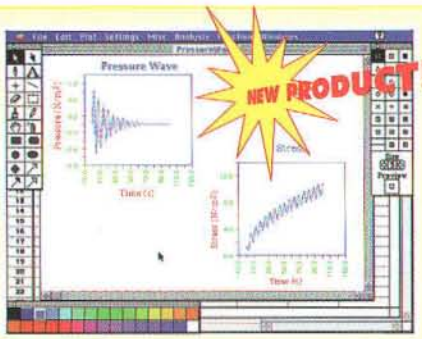
AppMaker

by Bowers Development

- Develop the user interface for a Macintosh application using the original interface builder
- Just point and click to design your application
- Creates resources and generates excellent source code
- Supports most development environments including Metrowerks, Symantec, or MPW; C, C++, or Pascal; procedural or object-oriented, using PowerPlant, TCL, or MacApp
- The generated code uses the Universal Headers to provide PowerMac compatibility
- Great tool for beginners to learn object-oriented and Macintosh Toolbox programming techniques
- Includes one-year subscription on CD and hardcopy documentation

(SAPPMAKE) Our Price **$199**

SuperAnalyst
by SuperSoft

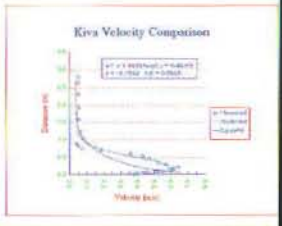


SuperAnalyst is an easy-to-use data analysis and plotting application written specifically for the Apple Macintosh computer and PowerPC. It provides a wide range of X-Y plotting and analysis capabilities at a click of the mouse. It is easy to use and provides interactive control of the appearance of almost every characteristic of your plot. You can overlay multiple plots on the same graph, and the number of points is limited only by the memory of you computer.

- 16 Plot Types: Scatter (Box and Cross), Line (with and without symbols), Function, Log-Log, Semi-Log, Double Y, Bars, Columns, Stacked Bars, Stacked Columns, Area, Pie, Polar, and Histogram.
- Function Plots: Plot mathematical equations you can input without constant reference to the manual. 24 intrinsic functions (cos, sinh, exp, ln, sqrt, erf, bes, gam, etc.).
- PLUS, set templates, reads many file formats, smooth, filter, and sort data, modify data, error bars, function and data integration, FFT tranforms, curve fits, etc.

(SSANAL) Our Price **$99**

SuperPlot
by SuperSoft



SuperPlot is a plotting library which can be linked and called directly from programs written in either Fortran, Pascal, or C. SuperPlot provides a simple way to plot data generated by your program, edit the plot, and then print, export, and/or save the plot for future reference.
A simple subroutine or procedure call hides your applications menu bar, puts up SuperPlot's menu bar, creates a new window, and plots your data. It provides you with complete control of the appearance of your plot using the mouse, menus, floating palettes and dialogs. Preparing presentation quality graphs of your data was never easier. SuperPlot runs on either the PowerPC or 68 K computers.

(SSPLOT) Our Price **$195**

NEW PRODUCT!

SuperPlotPRO
by SuperSoft

- Plotting data from your program was never easier
- A Plotting and Chart Library callable from Fortran, Pascal, and C
- Plot Types: Scatter, log-log, x semi-log, y semi-log, Ddouble Y, cross, line, line w/ symbols, bar, stacked bar, column, stacked column, area, pie, polar

(SSPLOTPRO) Our Price **$295**

QUED/M 3.0
by Nisus Software

- The programmer's text editor that defined the industry standard for speed and efficiency
- PowerPC native
- Features integrated support for Symantec C/C++, Metrowerks CodeWarrior 6, and MPW
- Supports all the major development environments on the Macintosh.
- Powerful editing features, including unlimited undo and redo, macro language, scripting, text folding, ten editable/appendable clipboards, markers, displaying text as ASCII codes, dynamic coloring of C/C++ keywords/comments, rectangular and non-contiguous selection
- Includes Celestin Company's APPRENTICE 4

(SQUEDM) Our Price **$89**

AG Author
by Lakewood Software

AG Author 1.0 is a full-featured Apple Guide authoring tool with fully customizable project template. The following features are unique to AG Author:

- Support for styled, colored, & hot text
- Fully customizable project template
- Flexible compile options
- Find & replace tool for scripts
- Multiple open projects
- Rapid deployment of project globals

(SAGA) Our Price **$99**

SEE RELATED PRODUCTS: AppleGuide Complete, Danny Goodman's AppleGuide Starter Kit, Real World AppleGuide

Web Ware
by BeachWare, Inc.

The ultimate collection of clip media and templates for building your own Web Page. An incredible selection of Shockwave movies, animated GIFs, buttons, bullets, dividers, and sample HTML pages. There are literally thousands of graphical elements on this disc, all there to spice up your web page. In all, it's about 300 megabytes of creativity only a mouse-click away! System Requirements: PC - 486 or better with 8 MB RAM, Sound card, SuperVGA, CD-ROM drive. Macintosh - Color Mac with 8 MB RAM, CD-ROM drive.

(SWEBW) Our Price **$24**

SoftPolish CD-ROM
by Bare Bones Software

- The essential tool for software quality assurance on the Macintosh
- Helps you identify inconsistencies with Apple's user interface guidelines, misspelled words, missing resources, and other mistakes
- Provides tools to put the finishing touches on software distribution packages prior to release
- Works independently of any programming language or environment
- Ideal for sanity checking software throughout the development process

(SSOFTPOL) Our price **$99**

VText
by Vivistar

VText is a C++ add-on library for Metrowerks' PowerPlant application framework. VText provides complete Macintosh text support including: greater than 32kb text, undo, drag and drop editing, AppleEvent scripting and recordability, full support for multibyte characters and inline input methods including Japanese and Chinese text, and full support for bi-directional script systems including Arabic and Hebrew.

- Full featured text engine for Metrowerks' PowerPlant
- Stylesets and rulersets with tabs
- Flexible object oriented C++ API
- Full undo and drag and drop editing
- WorldScript savvy including bidirectional and multibyte scripts with inline editing
- AppleEvent factored for scriptability and recordability

(SVTEXT) Our Price **$349**

OpenGL for the Macintosh
by Conix Graphics

OpenGL is the premier 3D graphics library that allows software developers the ability to develop high-quality, interactive 2D and 3D graphics applications. OpenGL can perform the following wide range of functions which will enhance the development of all graphics software:

- Geometric primitives (points, lines, and polygons)
- RGBA or color index mode
- Viewing and modeling transformations
- Texture Mapping, Lighting, Shading and Z Buffering
- Atmospheric Effects (fog, smoke, and haze)
- Alpha Blending (transparency)
- Antialiasing, Accumulation Buffer, Stencil Planes
- Display list or immediate mode
- Polynomial Evaluators (to support Non-uniform rational B-splines)
- Feedback, Selection, and Picking Raster primitives (bitmaps and pixel rectangles)
- Pixel Operations (storing, transforming, mapping, zooming)

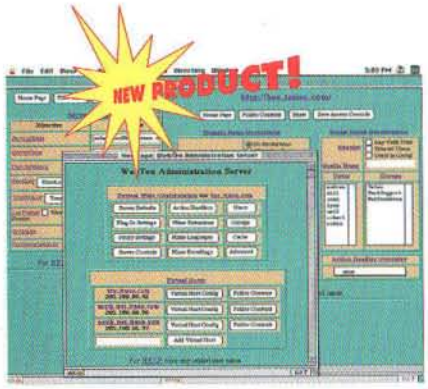
(SOPENGL) Our Price **$389**

WAIT... There's More!

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	OUR PRICE
Bee-one	SBEEONE	$139.00
C-tree Plus® Database Handler	SCTPDH	$895.00
CompileIt!	SCOMPIT	$149.00
CONIX PowerTools-10 Pack	SICPP10	$7,845.00
CPU Doubler	SCPU2X	$79.00
DesignWorks 4.0	SDWORKS	$995.00
dtF	SDTF	$695.00
EtherPeek	SEPEEK	$745.00
Fortran 77 SDK	SF77	$699.00
ICONIX PowerTools-6 Pack	SICPP6	$5,945.00
ICONIX PowerTools-8 Pack	SICPP8	$6,945.00
ICONIX PowerTools-AdaFlow	SICADA	$1,395.00
ICONIX PowerTools-ASCII Bridge	SICASCII	$1,395.00
ICONIX PowerTools-CoCoPro	SICCOCO	$1,395.00
ICONIX PowerTools-DataModeler	SICDATAMOD	$1,395.00
ICONIX PowerTools-FastTask	SICFASTTASK	$1,395.00
ICONIX PowerTools-FreeFlow	SICFREEFL	$1,395.00
ICONIX PowerTools-Object Modeler	SICOBJMOD	$1,395.00
ICONIX PowerTools-PowerPDL	SICPOWER	$1,395.00
ICONIX PowerTools-QuickChart	SICQUICKCH	$1,395.00
ICONIX PowerTools-SmartChart	SICSMART	$1,395.00
ICONIX Training & Consulting	TICONIX	$2,945.00
IMSL Math and Stat Library	SIMSLSTAT	$495.00
Info-Mac X	SINFOMAC10	$39.00
Ionizer Real-Time Spectral Reshaping Tool	SIONIZER	$800.00
LiveAccess™ 1 User Edition	SLAUE	$69.00
LiveAccess™ 1 Developer Edition	SLADE	$99.00
LiveCard	SLCARD	$149.00
LJ Profiler	SLJPROF	$295.00
MacFlow™: Flowchart Design and Development	SMACFLO	$179.00
Mac Source II	SMACSOURCE	$29.95
Nisus Writer 5.0	SNISUSW	$220.00
Plan & Track™: Project Planning and Management	SPLNTRK	$179.00
Phyla™: Object-Oriented Database	SPHYLA	$179.00
r-tree Report Generator	SRTRG	$445.00
Spellswell Plus 2.1	SSPELL	$49.00
Spyer	SSPY	$39.00
Visual Café	SVCAFEMAC	$199.00

WebTen
by Tenon Intersystems



WebTen is an industrial-strength, high-performance Apache Web server for Power Macs. WebTen's Web-based browser interface enables local or remote administration via your favorite browser. Since Apple's NeXT acquistion, Tenon has extended their unique "UNIX virtual machine" technology to produce a set of "Rhapsody-Ready" internet applications. WebTen is the first offering in this series.

- WebTen is the fastest Web server on Power Macintosh
- Sustains up to 10,000 connections a minute, or over 10 million connections a day
- Apache runs in Tenon's multi-threaded, pre-emptive multitasking environment
- Tenon's unique technology supports the widely acclaimed Apache Web server as a double-clickable Macintosh application

(SWEBTEN) Our Price **$495**

BBEdit 4.5
by Bare Bones Software

(SBBEDIT) Our Price **$119**

Also see Tools, Libraries and Utilities, page 8

webAlias 1.0
by Lakewood Software

webAlias 1.0 is an integrated image map editor and anti-aliasing text tool for web and graphic designers. Use webAlias to create complete web sites, single web pages, and graphic content for multimedia and web design projects. webAlias integrates support for line, shape, free form, field and button objects. webAlias' anti-aliasing features include support for embedded pictures and gradients in text, as well as multiple shadow and highlight effects.

(SWEBALS) Our Price **$129**

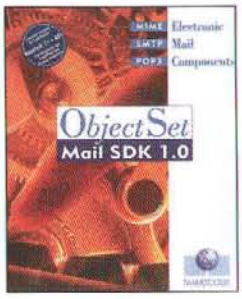
PageCharmer: Sizzling Effects...

PageCharmer 1.0
by Mainstay

PageCharmer is a set of customizable interactive applets that enhance web pages without writing a single line of HTML code. Whether the web site is already up and running or designing one from scratch, PageCharmer gives you the power to make it stand out from the crowd with sophisticated applets that can be personalized to fit most any need.

FEATURES:
LiveG-Map, LiveT-Map, LiveG-Button, LiveT-Button, LiveGT-Button, LiveG-Ticker, LiveT-Ticker, LiveG-Marquee, and LiveT-Marquee.

(SPGCHRM) Our Price **$99**



ObjectSet Mail SDK
by Smartcode Software

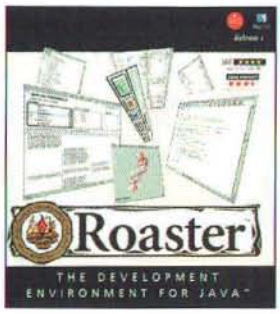
- Powerful C++ classes for integrating Internet e-mail in your applications
- Helps you write software that can share mail with other leading e-mail products
- Royalty-free MIME, SMTP, and POP3 APIs for Macintosh, Windows, and Unix
- Gives you the most robust MIME parser and encoder available
- Ideal for use in Internet and Intranet environments
- Comes complete with samples with documented, reusable source code
- Free standard technical support

(SOSMSDK) Our Price **$495**

HyperGuide 1.0
by Lakewood Software

HyperGuide 1.0 is a hybrid multimedia authoring tool and on-line documentation system for the Macintosh and World Wide Web. HyperGuide provides integrated searching, indexing and bookmarking features. Supported media elements include: rectangle and scrolling fields, lines and shape fills, most QuickTime-supported image formats, anti-aliased text and QuickTime VR movies. HyperGuide also includes an integrated screen capture utility and user-configurable slide show mode.

(SHYPGUD) Our Price **$149**



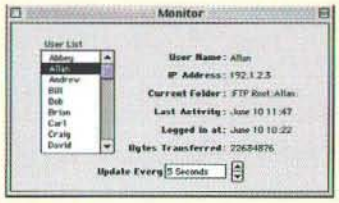
Roaster
by Roaster Technologies, Inc.

(SROAST3) Our Price **$99**

Also see Development Environments, page 5

One & 2-Day SHIPPING (call for details)

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.



Rumpus
by Maxum Development

Maxum's new, high-performance FTP server for the MacOS. Based on Maxum's RushHour TCP/IP implementation, Rumpus 1.0.1 offers the performance and reliability of high-end workstations with the ease of use, security, and flexibility of the Macintosh.

- Simplified setup, with no need to configure AppleShare, File Sharing, or Users & Groups for simple anonymous FTP
- Anonymous and/or secure server access, with separate security settings for anonymous vs. secure users
- Automatic MacBinary and Binhex encoding
- Complete logging, with separate anonymous and secure access logs, including anonymous user passwords
- Up to 32 simultaneous connections

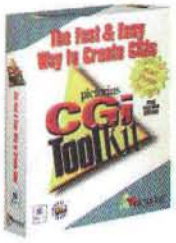
(SRUMP) Our Price **$195**

Power MachTen 4.0.3
by Tenon Intersystems

MachTen is the only Macintosh product that can turn your Macintosh into a complete Unix workstation. Based on BSD4.4 and the Mach kernel, MachTen brings the power of Unix to your desktop at an extremely attractive price point. MachTen enables you to:

- Run a high speed internet server, complete with WWW, FTP, NFS, DNS and print service
- Build a Mutihomed Web Server
- Develop applications in a Unix development environment, replete with the acclaimed GNU development toolset
- Program in Ada, C, C++, Pascal, Fortran, and more
- Run Xwindows applications, from remote workstations or on your Macintosh
- Run hundreds of Unix applications, already ported for MachTen and available on our Ported Applications CD-ROM
- Run Software.com Inc's acclaimed Post.Office mail transport service

(SM10PPC) Our Price **$695**



CGi Toolkit
by Pictorius, Inc.

The Pictorius CGi Toolkit is the fast and easy route to high performance CGIs and ACGIs for your Mac Web site.

- Interactively develop CGIs while the web server, the CGI Toolkit and the browser are running on the same machine
- Interactively develop, test and debug CGIs before compiling
- Powerful debugger allows you to edit code, roll back, code and change input values while your application is running
- Fully object oriented so you can re-use your code
- Automatic handling of Apple Events so you can concentrate on building functionality
- Easy creation of multi-function CGIs which reduces application footprint and RAM usage

(SCGITLKT) Our Price **$149**

WAIT... There's More!

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	OUR PRICE
OOFILE Reporter Writer	SOORW	$499.00
ScriptDemon	SSDEMON	$949.00
WebSiphon	SWSIPHON	$495.00

WindowScript

NEW PRODUCT!

by Royal Software, Inc.

WindowScript is the ultimate tool for designing Macintosh user interfaces using HyperCard. Design Real "Macintosh" user-interfaces right inside HyperCard. Until now you either created HyperCard stacks or Macintosh applications. With WindowScript you can literally bring the look and feel of a real Macintosh user-interface to HyperCard. If you're a HyperCard developer, interface designer, application developer, program manager or tester searching for a prototyping tool, WindowScript is perfect for the job.
(SWSCRIPT) Our Price **$149**

Script Debugger

by Late Night Software Ltd.

- A powerful and flexible AppleScript authoring tool – get the most from AppleScript!
- Advanced debugging environment offers single-step script execution with breakpoints
- Script Debugger dictionary browser features a graphical view of objects provided by scriptable applications
- Includes Late Night Software Scripting Additions – a collection of more than 70 new AppleScript commands, and Scheduler, a utility that allows you to launch scripts at pre-determined times

(SDEBUG) Our Price **$129**

Scripter 2.0

by Main Event Software

For professionals, for novices, for webmasters, for solutions providers, there's only one serious choice. Scripter!

- Scripter and FaceSpan work together: one click opens your FaceSpan script in Scripter, another sends it back
- Debug handlers without modifying your scripts using the Call Box
- Applet simulation, live editing, Object map, associated terminology
- Search backwards, block generators, more navigation shortcuts, more drad-and-drop, and an even more enhanced trace log
- Now Includes ScriptBase; stores your data and media elements and share them between scripts all with a special new browser
- Easily write and compile scripts that have handler declarations and other vocabulary specific to a particular scriptable application
- Scripter is the natural companion to AppleScript for users at all levels of proficiency. Don't write scripts without it!

(SSCRIPTER) Our Price **$199**

TCP/IP Scripting Addition

by Mango Tree Software

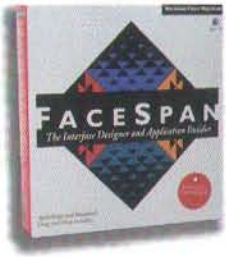
- Award-winning AppleScript scripting addition
- Allows you to write scripts using MacTCP™ commands in AppleScript™
- Send e-mail or files through a script, check if users are logged on (via Finger), automate FTP, Gopher, NetNews, Telnet, and LPR, verify links in HTML documents, and quickly write many other TCP/IP client-server programs
- Works with AppleScript, MacTCP 2.0.4 and Open Transport

(STCP) Our Price **$49**

FaceSpan 2.1 SRT

by Digital Technology International

Single-user version of the popular, cutting edge interface design tool which gives you the power to build and customize Macintosh applications quickly and easily. Used in conjunction with AppleScript or any other OSA language.
(SFACESRT) Our Price **$99**



FaceSpan v2.1

by Digital Technology International

- Develop integrated software, make stand alone applications, create friendly interfaces
- Develop quick prototypes, print multiple pages with sophisticated layouts
- Script essential elements of the FaceSpan
- Play and record sounds as either "snd" resources or as "AIFF" files
- Create miniature or complete apps that run on either Power PC or 68k computers
- Use precise time measurement for implementing timed behaviors — New properties
- Proportionally scale PICT images -Align images in a pictbox - Automate any application
- Monitor and respond to low-memory situations-Increased support for Frontier UserTalk!

(SFACESPAN) Our Price **$299**

DynaMorph 1.5

by Morph

DynaMorph

DynaMorph is the only cross-platform, server-side scripting language. Easily build and maintain dynamic websites and web-based applications. Access external databases, separate the format of a website from its content, conduct e-commerce transactions and more. DynaMorph makes sites and applications completely portable.
(SDYNA) Our Price **$399**

WAIT... There's More!

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	OUR PRICE
PreFab Player	SPLAYER	$95.00

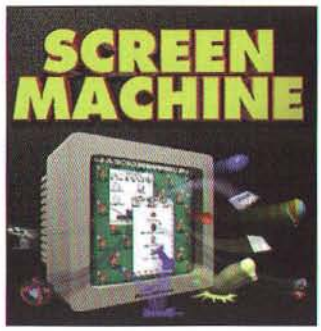
Clip VR™
by eVox Productions

Clip VR™ is a new digital image library offering high quality Photographic Virtual Reality (PVR) images for use with Quicktime® VR and other desktop VR tools.

Clip VR™ Panoramic Image components include alpha channel masks. Combine elements into a composite panorama which is converted to the finished QuickTime VR movie using Make QTVR Panorama Tool. The Components ("Clips") include complete 360 degree scenes, libraries of 360 degree terrains, 360 degree skies, buildings, and objects. Images are provided as .PICT files AND QuickTime VR movie files. Clip VR™ allows you to create high quality VR worlds from pre-photographed component images. Clip VR™ can be used to add excitement to a web site, to increase the interactive value of a CD-ROM, or simply for fun! Requires imaging editing program such as Adobe Photoshop™ to perform .PICT image compositing.

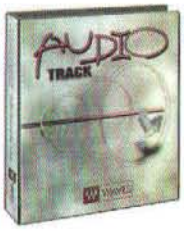
(SCLIPVR) Our Price **$89**

Screen Machine
by BeachWare, Inc.



Personalize your computer screen with this dynamic and useful collection of Screensavers and Wallpapers. Choose from over 100 original Screensavers such as flying airplanes, bouncing coffee cups, falling climbing gear, shifting psychedelic patterns, or floating spaceships. Customize your computer desktop with over 150 exciting new wallpapers such as sand, rocks, cloth, coins, cartoons, or unique patterns. Included is an easy to use browser program that lets you sample all of the Screensavers and Wallpapers before installing them on your computer.

(SSM) Our Price **$24.95**



AudioTrack
by WAVES

AudioTrack is a software plug-in for native processing on digital audio recording and editing systems. Waves AudioTrack combines the most-needed audio processors into a single piece of software, including 4 bands of equalization, compression/expansion, and noise gating. AudioTrack is ideal for preparing audio for InterNet streaming formats, processing individual mono/stereo tracks of audio and audio for video editing systems including digital sequencers.

Feature list

- A single window interface
- 4-band ParaGraphic Equalizer, Compressor/Expander and Gate
- Instantaneous A/B comparisons of on-line settings
- Pretested setup libraries supplied for various processing solutions
- Power Macintosh native processing (requiring no DSP board)
- Volume and gain reduction meters
- Peak hold and clip meters

Requirements:
The AudioTrack is compatible with all machines supported by Deck II, SoundEdit 16, Adobe Premiere 4.0 and Cubase VST 3.1

(SAUDIOTRK) Our Price **$270**

Captivate 4.6: Essential Graphics Utilities
by Mainstay

Captivate™ 4.6 is a powerful collection of graphics utilities for Macintosh, based on Mainstay's acclaimed screen capture utility, graphics and multimedia scrapbook, and graphics viewer. Captivate provides essential graphics utilities to the professional and hobbyist alike.

- Package includes: Captivate Select, Captivate View, and Captivate Store
- Any one of the three can be used alone, and together they make an unbeatable team
- Whether writing a training manual, creating an ad, or just creating a startup screen from your favorite picture, Captivate is everything professionals need at a price anyone can afford.

(SCAPTIV) Our Price **$79**

Media Cleaner Pro
by Terran Interactive

Use Media Cleaner Pro 2.0 to optimize and compress video for CD-ROM, kiosk, or the Internet. Media Cleaner Pro automates your work flow allowing you to get the highest quality video, faster and easier than any other program on the market.

- Includes Adobe Premiere Export module
- Optimal palette generation, Drag-and-drop batch processing
- RealMedia, VDOLive and improved QuickTime support
- Dynamic Preview Window, the Media Wizard, multiprocessor support and more!

System Requirements:
68040 Mac or better (PowerPC strongly recommended, req'd for RealMedia), QuickTime 2.0 or later (2.5 strongly recommended)
8 Mb application RAM, MacOS 7.0.1 (7.5 or later recommended)
SoundManager 3.2, CD-ROM Drive

(SMCP) Our Price **$359**

Registered owners of Movie Cleaner Pro 1.3 or earlier can upgrade

(SMCPUP) Our Price **$129**

KAIDAN
Digital Imaging Technology

NEW PRODUCT!

Magellan QC
by Kaidan

The Magellan QC is capable of handling objects as large as six inches in diameter and five pounds in weight, the Magellan QC is the perfect choice for those needing to capture small objects at a reasonable price. Real-world objects can be turned into 3-D virtual reality movies using the QuickTime VR Authoring Studio and the Magellan QC.

The Magellan QC leverages the capabilities of the Connectix™ Color QuickCam™ digital camera for QTVR object capture. The Color QuickCam's close focusing capability (one inch to infinity), 640 x 480 resolution, serial interface (no video card required), 24-bit color support, convenient size and low cost make it an ideal camera for many simple QTVR object movies. Using the Magellan QC is easy. Simply locate the object on top of the adjustable pedestal, perhaps with a small piece of double-sticky tape, and then adjust the arms and pedestal so that the center of the object is centered in line with the camera and the rotation axis of the swingarm.

(HMAGQC) Our Price **$299**

Magellan Accessories

Magellan QC Pedestal Set

Two extra pedestal tube assemblies, one 2.5" and another 6" long. These extra pedestal tubes are used to support objects of varying sizes on the Magellan QC.

(HMAGPED) Our Price **$39**

Magellan QC Detent Wheels

A pair of optional detent wheels (Color = Gold) with 8 (45 deg), 12 (30 deg), 14 (25.7 deg), 16 (22.5 deg) and 18 (20 deg) settings. The standard wheels (Color = Aqua) provide 10, 15, 20, 24 and 36 positions.

(HMDWHLS) Our Price **$74**

NEW PRODUCT!

QuickPan Magnum
by Kaidan

The QuickPan Magnum Series consists of two models, the QPX-1 and QPX-2. Featured on both models is the new QPU-2 camera bracket. Based on the highly successful KiWi, it provides a sturdy, collapsible system for the mounting and adjusting of a wide variety of cameras and camcorders. The new base designs used on the Magnums are a refinement of our earlier bases, with the QPX-1 having a fixed base and the QPX-2 having a new low-profile micro-tilt adjustment stage. The easily adjustable click-stops will let you capture a panorama in a few seconds. The QPU-2 has two accessories, a Landscape Bracket for positioning the camera in the landscape orientation (QPLB-1) and a Counterweighting Kit (QPCW-1) used to balance large cameras or camcorders, such as the Sony VX-1000, that have a center of mass well behind the pivot axis.

QuickPan Magnum-1 (HQPMAG1) Our Price **$499**
QuickPan Magnum-2 (HQPMAG2) Our Price **$549**

QuickPan Magnum Accessories

Counterweighting Kit

The Counterweighting Kit includes a weight and adjustable arm that is used to offset the weight of large, heavy cameras and camcorders.

(HWHTKT) Our Price **$129**

Detent Wheel

Detent Wheel (5-inch) (Color = Purple): 10, 14, 18, 24 and 30 Position (QPDD-2)

(HQDWHLS) Our Price **$49**

QuickTilt Leveler

A leveling stage, similar to the one found on our QuickPan Magnum QPX-2, that mounts between your panhead or camera and your tripod. It makes the leveling process quick and easy. Particularly useful when you plan to shoot a number of QTVR/VR nodes in a short period of time.

(HQTLVLR) Our Price **$149**

NEW PRODUCT!

KiWi
by Kaidan

The KiWi™ is the most affordable VR/QTVR panhead, bringing digital photographic panoramas to an even wider audience. It's the perfect companion to programs such as QuickTime VR Authoring Studio, PhotoVista and Nodester, providing a complete solution for anyone interested in adding VR panos to their websites and multimedia applications.

The KiWi™ consists of two intersecting black anodized aluminum struts that adjust and lock to accommodate a wide range of cameras, such as the Apple QuickTake 100/150/200, Kodak DC50/120, APS film cameras and 35mm SLRs equipped with wide-angle lenses. KiWi™ attaches to any standard tripod and camera equipped with a standard 1/4-20 mounting thread.

(HKIWI) Our Price **$99**

One & 2-Day SHIPPING (call for details)

NEW PRODUCT!

KiWi+
by Kaidan

The KiWi+ adds a compact, yet durable click-stop mechanism and the same twin-axis bubble level found on the top-of-the-line QuickPan Magnum Series heads. The twin-axis bubble level (recommended by Apple and VR professionals) provides a clear indication of level, even when the unit is slightly above eye level. The click-stop mechanism uses easily replaceable detent discs, which are available in a number of positions (8, 12, 16, 18, 20). The KiWi+ ships with one disc of your choice and extra discs are available separately or as a set. The click-stops speed the process of shooting a panorama by eliminating the need for the photographer to look at the unit in order to visually align the index increment.

(HKIWIP) Our Price **$249**

KiWi and KiWi+ Accessories

QuickTilt Leveler

A leveling stage, similar to the one found on our QuickPan Magnum QPX-2, that mounts between your KiWi or KiWi+ and your tripod. It makes the leveling process quick and easy. Particularly useful when you plan to shoot a number of QTVR/VR nodes in a short period of time.

(HQTLVLR) Our Price **$149**

KiWi-to-KiWi+ Upgrade

Includes the necessary parts required to turn your KiWi into a KiWi+ — adding click-stops and the twin-axis bubble level. Comes with a detent disc of your choice (8, 12, 16, 18 or 20 positions).

(HKIWIUP) Our Price **$199**

KiWi+ Detent Discs

KiWi+ Detent Discs are available singly or in a set of four. In both cases you get to choose whichever discs you need.

(HDTDISC) Our Price **$24.95** each

(HDTDISC4) Our Price **$89** set of four

Choices include: 8, 12, 16 ,18, or 20 position detent disc

Landscape Bracket

The Landscape Bracket is a right angle bracket that allows you to mount the KiWi or KiWi+ upright camera bracket in a horizontal orientation. This bracket is primarily used for cameras that have a limited field of view and you need to limit the number of shots.

(HLDBRAC) Our Price **$42**

Flash Hotshoe Level

A dual-axis bubble level that slides into your camera's hotshoe. It's a useful tool to help level the camera on the upright camera bracket.

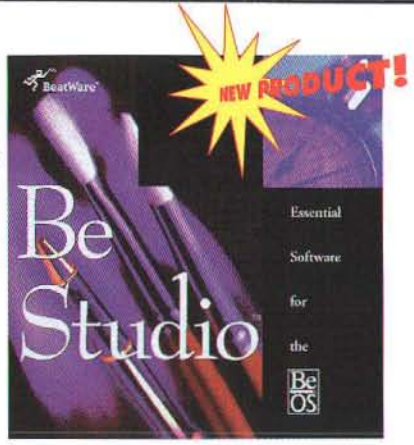
(HFLASH) Our Price **$39**

Offset Spacer

The Offset Spacer is a circular spacer that may be required for very narrow cameras (i.e. certain Ricoh digital cameras) in order to position the center of the lens over the pivot axis.

(HOFFSPAC) Our Price **$24.95**

Be Studio
by BeatWare



Developed specifically for the BeOS, Be Studio is the first graphics application to offer full multithreading for unbelievable responsiveness and unbeatable speed. Be Studio includes Paint for editing photos or creating original artwork and Draw, a vector-based tool for drawing crisp designs and prints. Be Studio offers the BeOS developer the ability to:

- Design application icons quickly and easily
- Edit and export images easily to the World Wide Web
- Manipulate several images at once without degrading performance
- View updates from a variety of perspectives simultaneously
- Import and Export images as GIF, JPEG, PNG, TIFF, and PNM files
- Plug-in third party tools and filters

Be Studio requires the BeOS, 16 MB of RAM (32 recommended), 3MB available hard disk space and a 256-color display adapter. Price includes all major and minor upgrades through version 2.0 via electronic distribution.

(SBESTUD) Our Price **$99**

Order Toll-free
800-MACDEV-1
(800-622-3381)

webAlias 1.0
by Lakewood Software

webAlias 1.0 is an integrated image map editor and anti-aliasing text tool for web and graphic designers. Use webAlias to create complete web sites, single web pages, and graphic content for multimedia and web design projects. webAlias integrates support for line, shape, free form, field and button objects. webAlias' anti-aliasing features include support for embedded pictures and gradients in text, as well as multiple shadow and highlight effects.

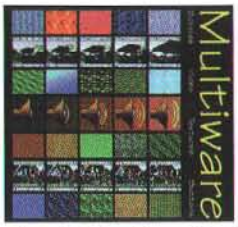
(SWEBALS) Our Price **$129**

Music Tracks
by BeachWare, Inc.

A new PC/Mac & Audio multimedia music CD-ROM. The clips include musical introductions, fanfares, background music, and more. This collection offers you 100 music clips stored in .WAV format for Windows, SoundEdit & AIFF formats for Macintosh and as Audio tracks for audio CS players. All of the music clips are completely license and royalty-free!! Mac System requirements: Mac Plus or greater, CD-ROM drive. PC system requirements: Windows 3.1 or later, Sound Blaster compatible board, CD-ROM drive.

(SMT) Our Price **$24.95**

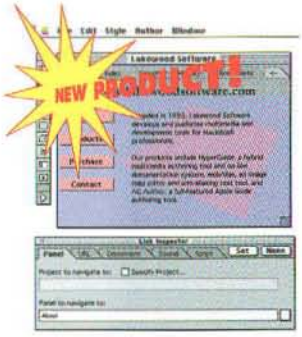
MultiWare Multimedia Collection
by BeachWare, Inc.



Introducing a new Audio multimedia music CD-ROM for the Macintosh. This disc is a collection of clips ideal for Desktop Presentations and other Multimedia applications. This incredible collection of license-free media clips is bursting with 240+ color pictures and backdrops (PICT), 200+ sound & music clips (SoundEdit), 140+ QuickTime movies, and a variety of multimedia tools for use with the Macintosh.

(SMWMC) Our Price **$24.95**

HyperGuide 1.0
by Lakewood Software



HyperGuide 1.0 is a hybrid multimedia authoring tool and on-line documentation system for the Macintosh and World Wide Web. HyperGuide provides integrated searching, indexing and bookmarking features. Supported media elements include: rectangle and scrolling fields, lines and shape fills, most QuickTime-supported image formats, anti-aliased text and QuickTime VR movies. HyperGuide also includes an integrated screen capture utility and user-configurable slide show mode.

(SHYPGUD) Our Price **$149**

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	OUR PRICE
AudioTrack	SAUDIOTRK	$270.00
Captivate 4.6 Essential Graphics Utilities	SCAPTIV	$79.00
Media Cleaner Pro	SMCPUP	$359.00

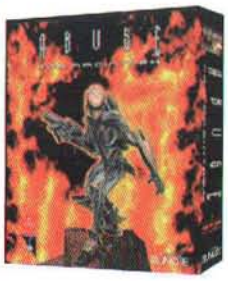
Power3D

by Techworks

The power of an arcade on your PowerPC Based on award winning 3Dfx VooDoo Graphics. The Power3D works with your existing graphics card and multi-sync monitor to provide you the absolute in 3D performance. Install the Power3D in your PowerPC (requires one available PCI slot in your system) and use the provided pass-through cable to turn your PowerPC into a Power Arcade system!
Power3D comes bundled with these awesome 3D enabled games:

- Quake®: Episode 1 (8 level) by Id Software
- MechWarrior® 2 by Activision
- VR Soccer™ by VR Sports (Actua™ for Europe)
- Weekend Warrior™ by Bungie

(SPWR3D) Our Price **$249**



Abuse

by Bungie Software

Abuse is 360 degrees of side-scrolling action. Run, jump, fall and fly in any direction - through industrial corridors, caverns and sewers. Destroy enemies in any direction with grenade launchers, rocket launchers, napalm and nova spheres! Avoid deadly traps with jet packs and turbo boost!
Key Features:

- Point and Kill Interface. Move and annihilate mutants in complete 360° freedom
- Blast your way through floors, walls and ceilings in search of the ultimate power-up!
- Abuse is 360 degrees of side-scrolling action. Run, jump, fall and fly in any direction – through industrial corridors, caverns and sewers

(SABUSE) Our Price **$51**

1000 Games for Macintosh

by BeachWare, Inc.

The best Macintosh game disc in the entire world, this CD-ROM contains over one thousand great shareware and public domain programs. Battle ugly aliens, blast apart run-away asteroids, deal yourself that royal flush or solve that 3-D puzzle, this disc has it all! System requirements: Mac Plus or greater, CD-ROM drive, and 2 MB of available RAM (4 MB of RAM when running under System 7).
(STGM) Our Price **$24**

Myth The Fallen Lords

by Bungie Software

***PC GAMES MAGAZINE'S*, Most Anticipated New Game Award**

The Fallen Lords is a fully 3D real-time strategy game of epic battle. A multimetric game, Myth: The Fallen Lords gives gamers unprecedented freedom to view their forces, orbiting around the heads of a formation or zooming in for a close-up on savage melee. Myth: The Fallen Lords includes maps designed for networking, and alternate networking scenarios like Assassin and King of the Hill.
(SMYTH) Our Price **$49**

Classic Arcade

by BeachWare, Inc.

Ten of your favorite coin-arcade games, redone with killer graphics and sounds! Walk through a virtual arcade and test your game playing skills with these exciting arcade classics. Ten games, including Moon Lander, Astro-Boing, Hyper Hockey, Ballistic Avenger, and more. System Requirements: Mac — Color Mac with 8 MB RAM, CD-ROM drive. PC 486 with 8 MB RAM, Sound card, SuperVGA, CD-ROM drive.
(SCLA) Our Price **$24**

Marathon Trilogy Box Set

by Bungie Software

The Marathon Trilogy Box Set brings all three Marathon games together in one affordable package, with tons of extras thrown in. Besides Marathon, Marathon 2: Durandal and Marathon Infinity, you'll also receive a staggering 1200 maps, featuring never-released Bungie maps and the winners of the Infinity Mapmaking Contest, The Marathon Scrapbook (a behind-the-scenes look at themaking of the Marathon games), Marathon collectables like the Marathon 3-sticker set, and to top it off, the award-winning game that laid the groundwork for Marathon: Bungie's breakthrough Pathways Into Darkness. The Marathon Trilogy Box Set is native to the Power Macintosh, utilizes the graphics acceleration of 630 and 6200 machines, is 8, 16 and 24-bit color capable, and can be played with joysticks and game pads. The package requires a 68040 or higher Macintosh, CD-ROM drive, 8-bit color monitor (13" recommended), and System 7 or later.
(SMTBS) Our Price **$65**

WAIT... There's More!

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	OUR PRICE
A Zillion Sounds	SAZS	$24.00
Casino!	SCAS	$24.00
Night Sky Interactive	SNSI	$24.00
Trivia Warehouse 2000	STW2K	$24.00

metrowerks

GeekWare™
You live it, you breath it... you might as well wear it! (XL only)

Alien T-Shirt
(AALIEN) Our Price **$9.95**

Arnold T-Shirt
(ACWARNLD) Our Price **$9.95**

Arnold Jr. T-Shirt
(AARNOLDJR) Our Price **$9.95**

Blood, Sweat & Code T-Shirt (SS)
(ACWSBLOOD) Our Price **$9.95**

Blood, Sweat & Code T-Shirt (LS)
(ACWLBLOOD) Our Price **$14.95**

CodeWarrior Baseball Cap - Black
(ACWBHAT) Our Price **$14.95**

CodeWarrior Sweatshirt - Black
(ACWSWEAT) Our Price **$29.95**

CodeWarrior Hawaii Five-0 Shirt
(ACWHAWAII) Our Price **$9.95**

CodeWarrior Winter Hat
(AWINHAT) Our Price **$14.95**

Debugger Boxer Shorts
(ADBOXER) Our Price **$16.95**

Discover Programming T-Shirt
(ADISCPT) Our Price **$9.95**

LEGTOP PODEUM™

Podeum Sport
by Rach, Inc.

Now you can have laptop stability, drop protection & mobility. If you're looking for an inexpensive, non-technical gift for a laptop owner – look no further.

- Strap your laptop to your leg
- Universal size - fits all laptops and all legs (13" - 46")
- Velcro Velcoins attach your laptop to the podeum
- Podeum allows working angles up to 90 degrees
- Dropped laptops account for 41% of all laptop fatalities
- Manufacturer's lifetime warranty.

(APODSP) Our Price **$39**

MacTech® Mouse Pad

Slide on this! With an extra-large surface (11" by 10") and a deluxe sleek plastic coating, you'll be zooming across your screen in no time at all. Speed limit not enforced!

(AMTPAD) Our Price **$8.95**

For Macintosh Programmers & Developers

MacTech®
MAGAZINE

MacTech® Magazine

MacTech keeps Mac programmers & developers up to date with everything they need to know about software development. Topics like Rhapsody, Java, QuickTime, OPENSTEP, Objective-C, C/C++, Object Oriented Technologies, product reviews and much more!

Subscriptions:
(MTYRDM) US/Domestic for 12 issues **$47**
(MTYRCM) Canadian for 12 issues **$59**
(MTYRFM) International for 12 issues **$97**
Back Issues: each plus shipping (subject to availability) **$10**

MacTech® CD-ROM Volumes 1-12

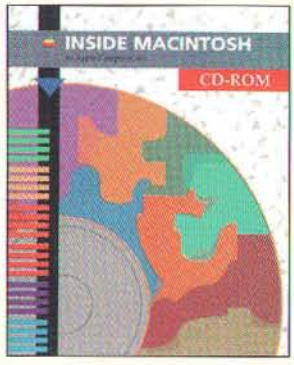
- Includes Apple's *develop* issues 1-29 (1990-1997)
- Almost 1600 articles from all 139 issues of MacTech Magazine (1984-1996) and through may of 1997
- Improved hypertext, improved indices, and a new THINK Reference Viewer – for lightning quick access!
- New hyperlinks between articles
- 100+ MB of source code — use them in your applications, with no royalties!
- Full version of THINK Reference™ — the original online guide to Inside Macintosh, Vols. I-VI
- 80MB of FrameWorks/SFA archives and the most complete set of FrameWorks archives known
- Sprocket™! MacTech's tiny framework that compiles quickly and supports System 7.5 features
- The best threads from the Macintosh programmer newsgroups plus thousands of notes, tips, snippets, and gotchas
- Popular tools that Macintosh programmers use to increase their productivity and much more!

(SMTCD12) Volumes 1-12 Our Price **$129**
(SMTCD12U) Upgrade from any previous version Our Price **$49**

INSIDE MACINTOSH
by Apple Computer, Inc

Inside Macintosh: CD-ROM

by Apple Computer, Inc.
More than 25 volumes in electronic form. Includes: QuickDraw™ GX Library, Macintosh Human Interface Guidelines, PowerPC System Software, Macintosh Toolbox Essentials and More Macintosh Toolbox, QuickTime and QuickTime Components. Access over 16,000 pages of information with Hypertext linking and extensive cross referencing.
(BIMCD) Our Price **$89**



Order Toll-free
800-MACDEV-1
(800-622-3381)

New Books!

The Official BBEdit Book

by Bob LeVitus and Natanya Pitts

The Official BBEdit Book makes it easy for today's Webmasters to speed their own interactive development projects using this powerful editing environment. Bare Bones incorporates features like floating palettes, HTML support, and syntax coloring to enhance an already extensive feature set, making BBEdit a leading Web authoring tool. Super-fast text processing, easy scripting, wide extensibility, strong GREP-style search-and-replace capabilities, and support for 13 languages make it easy to see why so many Web developers use BBEdit as their primary authoring tool.

(BOFBB) Our Price **$35**

Effective C++, Second Edition:

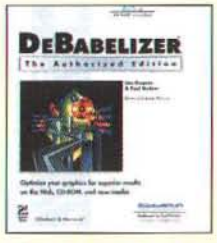
50 Specific Ways to Improve Your Programs and Designs

by Scott Meyers

Effective C++, 2nd Edition includes: Expert guidance on object-oriented design, class design, and the proper use of inheritance

- An examination of the standard C++ library, including how the Standard Template Library and classes like string and vector affect the structure of well-written programs
- Discussions of late-breaking language features like in-class constant initializations, namespaces, and member templates
- Wisdom usually possessed by only the most experienced developers
- Effective C++ continues to be essential reading for every developer working with C++.

(BEFFC) Our Price **$34**



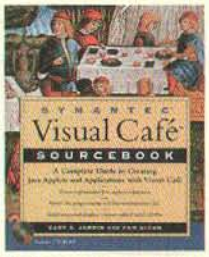
DeBabelizer

by Lise Despres and Paul Vachier

DeBabelizer: The Authorized Edition is the official guide for Web designers, multimedia creators, artists and production specialists who want to take advantage of this powerful tool.

- Create graphics and images for the Web that download fast and look amazing
- Optimize graphics for CD-ROM, video, and animation
- Optimize and manage colors using the SuperPalette(TM)
- Discover DeBabelizer tips and advice from industry experts
- Master basic manipulation techniques, including rotation, scaling, cropping, and text overlay
- Explore key production techniques in all areas of graphics processing

(BDEBTAE) Our Price **$40**

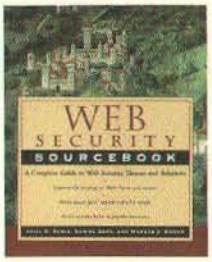


Symantec Visual Cafe Sourcebook

by Cary A. Jardin and Pam Dixon

Symantec Visual Cafe, the first visual Java development tool that gives programmers a sophisticated set of tools. This book teaches programmers how to use Symantec Visual Cafe to create Java applets. It provides a thorough introduction to the language and gives advanced Java programmers information on how to use Visual Cafe to create their own Java development tools.

(BSYMSOUR) Our Price **$35**



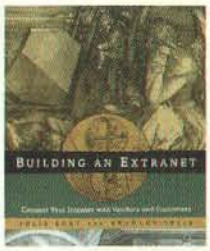
Web Security Sourcebook

by Aviel D. Rubin, Daniel Geer and Marcus J. Ranum

Technical tools and techniques for building secure web sites and applications

This book shows web masters, web managers, and web designers the hands on programming techniques necessary to build secure web sites. Readers will learn how to secure the server, use firewalls and cryptography, write secure Java applets and CGI scripts and more. Companion Web Site includes source code examples plus updates on the latest security threats and techniques.

(BWEBSER) Our Price **$26.99**

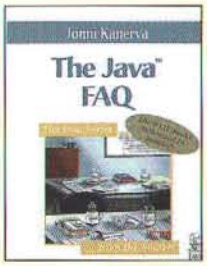


Building An Extranet: Connect Your Intranet With Vendors And Customers

by Julie Bort and Bradley Felix

Building an Extranet will help companies use their Intranet capabilities to supply information to selected customers and vendors through a secure Extranet. This book provides complete information and working details for building the network behind Intranets and Extranets, designing the applications, and getting everything up and running.

(BBLDEXT) Our Price **$26.99**



The Java FAQ

by Jonni Kanerva

Java FAQ provides an insider's view of the Java™ technology by posing and answering the most important, frequently asked questions about the Java programming language, Java applets, and Java stand-alone applications. The Java FAQ is unique in that it draws from the tens of thousands of questions sent to <java@java.sun.com> and provides authoritative answers direct from the creators of the Java programming language at JavaSoft.

(BJFAQ) Our Price **$25.13**

APPLE ENTERPRISE SOFTWARE

Getting Started With WebObjects
by Apple Enterprise Software

If you're a first time user, start here to learn the basics of how to create and run WebObjects applications.
(BGSWO) Our Price **$14**

WebObjects Developer's Guide
by Apple Enterprise Software

A guide to building and understanding WebObjects applications. Takes a close look at the WebObjects scripting language. Additional sections explain the WebObjects architecture and tells you how to integrate your code into the request-response loop, create reusable components, create client-side components, take advantage of powerful Foundation Framework features, and more. Filled with example code. For all WebObjects programmers.
(BWODG) Our Price **$16**

D'OLE Developer's Guide
by Apple Enterprise Software

Distributed OLE is available today, and this book shows you how to use it. Using real-world examples, the book steps you through the process of making your OPENSTEP objects available on Windows through OLE.
(BDOLEDG) Our Price **$22**

Discovering OPENSTEP, Mach
by Apple Enterprise Software

Introduces programmers to NeXT's OPENSTEP 4.0 Developer product by guiding them through the creation of three applications of increasing complexity. The tutorials demonstrate and explain programming techniques, Objective-C fundamentals, common APIs, and usage of the developement tools. Along the way they present summaries of important concepts and paradigms. The book also includes a chapter directing readers to programming resources, further information, and services such as training and support. An appendix offers a concise discussion of object-oriented programming.
(BDOSTEPM) Our Price **$15**

Discovering OPENSTEP, Windows
by Apple Enterprise Software

Discovering OPENSTEP provides an introduction to OPENSTEP programming on Windows NT. It guides the reader through the creation of three applications of increasing complexity. Along the way, it explains concepts and illustrates aspects of Objective-C, OPENSTEP classes, the development environment, and programming techniques. A short appendix offers a summary of object-oriented programming.
(BDOSTEPW) Our Price **$16**

Object-Oriented Programming and Objective C
by Apple Enterprise Software.

An introduction to the principles of object-oriented programming in OPENSTEP and the official description of the Objective-C language. Objective-C is easy to learn and use because it adds very little syntax to the C programming language. It's dynamic nature allows you to accomplish things not possible in most other object-oriented languages. For any OPENSTEP programmer.
(BOOPOC) Our Price **$24**

Working w/ Interface Builder (for EOF)
by Apple Enterprise Software

A hands-on, award-winning book designed to help you get your job done with the updated Interface Builder, released with NEXTSTEP 3.3 and the Enterprise Objects Framework 1.1. For any programmer using Interface Builder to design objects that truly work in NEXTSTEP.
(BWIB) Our Price **$24**

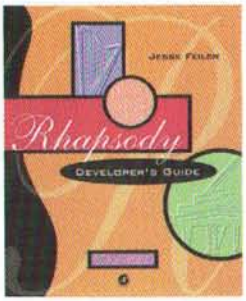
Using EOF 2.1 w/ OPENSTEP (Mach & Windows)
by Apple Enterprise Software

Using Enterprise Objects Framework with OPENSTEP describes how to create an Enterprise Objects Framework application on OPENSTEP. It includes a tutorial and a chapter on creating a user interface for an OPENSTEP Enterprise Objects Framework application.
(BUEOFO) Our Price **$14**

EOF Developer's Guide for EOF 2.1 (Mach & Windows)
by Apple Enterprise Software

The Enterprise Objects Framework Developer's Guide describes how to develop database applications using the Enterprise Objects Framework tools and classes. It includes an architectural overview of the product, and descriptions of programming tips and techniques. An appendix offers a summary of Entity-Relationship Modeling.
(BEOFDG) Our Price **$24**

EOF Developer's Guide for EOF 2.0 (BEOFDG20) Our Price **$24**
EOF Developer's Guide for EOF 1.x (BEOFDG1X) Our Price **$24**

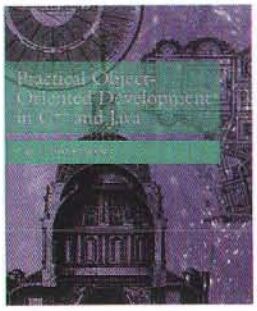


Rhapsody Developer's Guide

by Jesse Feiler

Covers the basic architectural principles of Rhapsody: the Mach microkernel, object-oriented programming, and the elements of a modern OS such as preemptive multitasking, protected memory, and symmetric multiprocessing. Also shows ways of getting to this new environment — Objective C, conversion tools, and the integration of Java — to develop Rhapsody products. Paperback, 450 pages.

(BRDG) Our Price **$35**

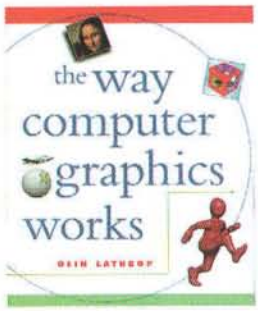


Practical Object-Oriented Development in C++ and Java

by Cay S. Horstmann

This book offers advice on real-world ways to use these powerful programming languages and techniques. Using the Unified Modeling Language (UML) methodology, expert Cay S. Horstmann gives you clear, concise explanations of object-oriented design, C++, and Java in a way that makes these potentially daunting operations more accessible than they've ever been before.

(BPOOD) Our Price **$31**



The Way Computer Graphics Works

by Olin Lathrop

A complete guide to mastering computer graphic basics. It is written in a frank, down-to-earth style covering everything from how computer graphics are different from fine art and photographs, to modeling, pixels, and the principles of animation. All of this is done without resorting to mind-numbing equations and impenetrable technical jargon.

(BWCGW) Our Price **$29.65**

WebMaster in a Nutshell, Deluxe Edition

by O'Reilly & Associates, Inc.

Includes CD-ROM!

Cross-platform, completely portable, and lightning fast, the CD-ROM is an invaluable addition to the webmaster's toolbox. The CD-ROM contains the Web Developer's Library — the full text of the latest editions of five popular O'Reilly titles: "HTML: The Definitive Guide, 2nd Edition"; "JavaScript: The Definitive Guide, 2nd Edition"; "CGI Programming on the World Wide Web"; "Programming Perl, 2nd Edition"; and "WebMaster in a Nutshell." The Deluxe Edition also includes a printed copy of "WebMaster in a Nutshell," the all-inclusive quick reference that belongs next to every webmaster's terminal. Includes CD-ROM & 356 page book.

Requirements: The CD-ROM is readable on all platforms, but requires a web browser that supports HTML 3.2, Java, and JavaScript.

(BWMNUTD) Our Price **$62**

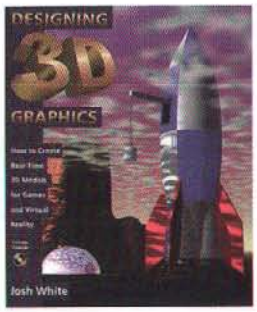
Debugging Macintosh Software with MacsBug

Includes CD-ROM!

by Konstantin Othmer and Jim Straus

MacsBug, from Apple Computer, Inc., is the leading debugging software program for the Macintosh. This book/disk package is an all-in-one kit for using MacsBug. Chapter 1 introduces MacsBug and describes the contents of the rest of the book. Chapter 2 describes how to install MacsBug and enough low level details about the Macintosh so that you can use MacsBug. Includes MacsBug 6.2 on disk.

(BDMSWM) Our Price **$31**



Designing 3D Graphics

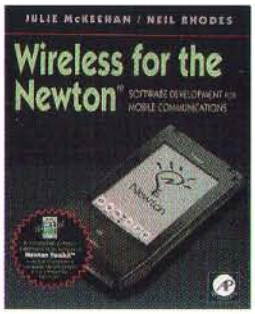
Includes CD-ROM!

by Josh White

In this powerful book/CD-ROM package, top computer graphics artist Josh White tells you everything you need to know to create sophisticated real-time 3D graphics for computer games and virtual reality. This book contains the in-depth knowledge of software tools and hands-on modeling techniques that Josh White has learned while creating artwork for over 20 commercial games, including Descent, Zone Raiders, Locus, Legoland, and others.

(BD3DG) Our Price **$35**

Order Toll-free
800-MACDEV-1
(800-622-3381)



Wireless For The Newton

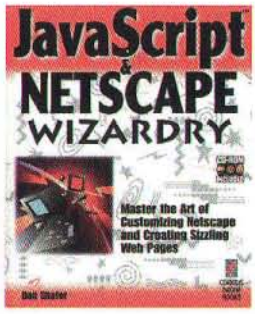
Includes Disk!

by Julie McKeehan and Neil Rhodes

A book that picks up where Programming for the Newton left off, teaching the reader how to develop Newton software on the Macintosh. The enclosed floppy disk provides a sample application, as well as a fully functional demonstration version of Newton Toolkit.

- Learn to develop Newton software on the Macintosh
- Hands-on Newton environment training with sample code
- Includes disk with sample source code for a Newton application, as well as demonstration NTK – the complete development environment for the Newton

(BWIRELESS) Our Price **$31**



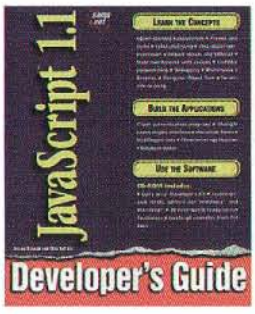
JavaScript & Netscape Wizardry

by Dan Shafer

The perfect book to show you how to turn Netscape into your own personal, customized operating system. Provides the inside tips and techniques for making your Web pages much more attractive. Shows you how to use all of the key features of the JavaScript language, including objects, methods, properties, events, and much more. Includes CD-ROM with numerous interactive scripts written in JavaScript you can add to your Web pages today. A complete set of the best Java applets. Useful plug-ins designed to supercharge Netscape and resources to help JavaScript programmers.

Includes CD-ROM!

(BJNWIZ) Our Price **$31**



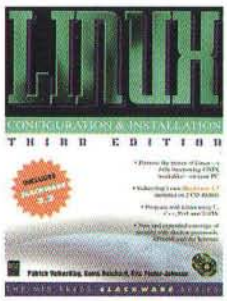
JavaScript 1.1 Developer's Guide

by Arman Danesh and Wes Tatters

Written by developers for developers. An advanced guide to creating professional Web applications with JavaScript 1.1 as deployed in Netscape Navigator 3.0, Microsoft Internet Explorer 3.0, and LiveWire. Includes CD-ROM with Sun's Java Developer's Kit, JavaScript and HTML Editors for Windows and Macintosh, 20 contributed ready-to-run JavaScripts and JavaScript examples from the book.

Includes CD-ROM!

(BJSDG) Our Price **$44**



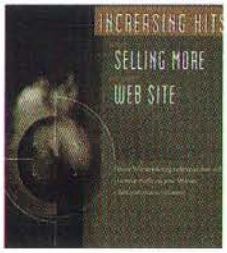
Linux Configuration & Installation, 3rd Edition

by Patrick Volkering, Kevin Reichard, and Eric F. Johnson

Linux, the leading UNIX variant, has garnered loads of attention within the UNIX community. The amazing thing about Linux is that you don't need a workstation to run it. Linux Configuration & Installation, Second Edition lets you run Linux today. Program with Linux using C, C++, Perl, and Tcl/Tk. The 2 CD-ROM pack offers one of the most popular Linux distributions, Slackware 96, and comes directly from Patrick Volkering, the creator of Slackware.

Includes CD-ROM!

(BLCI2) Our Price **$35**

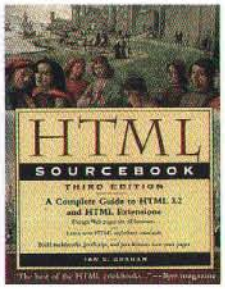


Increasing Hits and Selling More on your Web Site

by Greg Helmstetter

Written especially for entrepreneurs, corporate marketing managers, small business owners, and consultants, this valuable guide gives you rare tips and tricks you need to know to make your site a commercial success.

(BIHSMWS) Our Price **$22.45**



HTML Sourcebook, 3rd Edition

by Ian S. Graham

Critics everywhere agree, HTML Sourcebook is the best guide to HTML for Web professionals. That's because no other book makes it so easy for you to quickly master all the commands, tools, and expert techniques you need to create cutting-edge Web page documents. Completely revised and expanded by nearly 50 percent, this new third edition of the best-selling guide to HTML gives you the complete lowdown on all the changes and enhancements to the HTML, HTTP, and URL standards.

(BHTMLS) Our Price **$26.95**

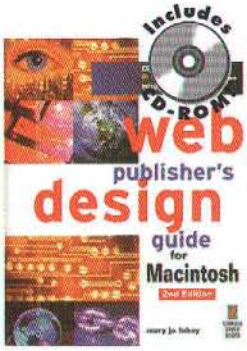
Teach Yourself Java for Macintosh in 21 Days

Includes CD-ROM!

by Laura Lemay and Charles L. Perkins with Timothy Webster

Add interactivity and multimedia to Web pages! A step-by-step guide to make your Website come alive. Learn the basics of programming Java applets and the concepts behind the Java language. Includes CD-ROM with a limited version of Roaster, the first commercial, integrated applet development environment for Java for the Macintosh!

(BJAVAMAC) Our Price **$36**

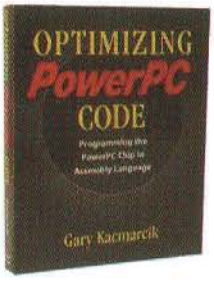


Web Publisher's Design Guide for Macintosh, 2nd Edition

by Mary Jo Fahey

This is the only book that takes you step-by-step through real projects designed by talented new media artists. Internet design experts share their design secrets and art files (look for art files on the companion CD-ROM by artist's last name). This expanded new edition includes Photoshop, Illustrator and DeBabelizer tricks from the first edition plus countless new ways to liven up your Web pages with 3D graphics, sound, movies, and much more.

(BWPDG2) Our Price **$35**

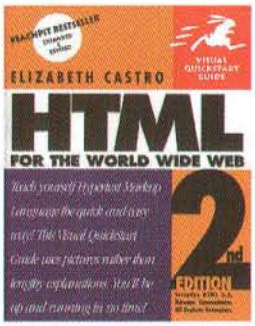


Optimizing PowerPC Code: Programming the PowerPC in Assembly Language

by Gary Kacmarcik

Take full advantage of the potential of the PowerPC by mastering the Assembly Language techniques. Learn to produce faster more robust software!

(BOPTPPC) Our Price **$35**

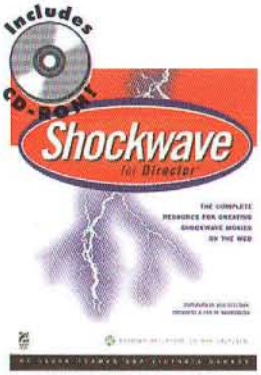


HTML For The World Wide Web, 2nd Edition

by Elizabeth Castro

Teach yourself Hypertext Markup Language the quick and easy way! This Visual QuickStart Guide uses pictures rather than lengthy explanations. You'll be up and running in no time. If you need to learn HTML fast – this is book is for you.

(BHTMLW2) Our Price **$16.15**

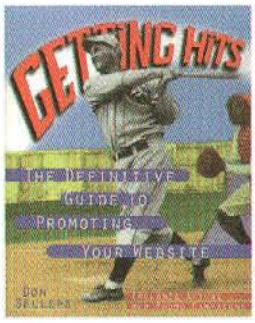


Macromedia Shockwave for Director

by Jason Yeaman and Victoria Dawson

The complete resource for creating Shockwave movies on the Web. This hands-on reference makes it easy to create Shockwave movies and put them on the Web. Expert tips from the creators of Macromedia's first Shockwave movies, together with detailed examples and instruction, provide everything you need to get started. Includes CD-ROM.

(BMSFD) Our Price **$27**

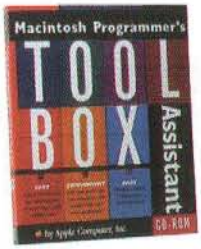


Getting Hits–The Definitive Guide To Promoting Your Website

by Don Sellers

Getting Hits explains in easy-to-understand language the underlying concepts behind the art of Web site promotion. Just a few of the topics you'll learn include: using search engines with URL's; finding related Internet groups or lists; understanding the nuances of click throughs and ad rates; and distributing press releases to key Internet contacts. With this book, you'll go beyond the conceptual and actually follow real-world tested promotional campaign strategies. Bring the world to your Web site!

(BGHITS) Our Price **$17.95**

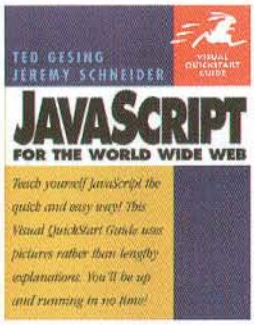


Programmer's Toolbox Assistant CD-ROM

Instant electronic access to Inside Macintosh essentials.
by Addison-Wesley Publishing

Get quick access to reference pages for over 4,000 Toolbox calls in your system software from their development environment. Essential information for Macintosh software developers. Hypertext links allow programmers to view related topics easily. The ultimate electronic reference tool for Macintosh programmers.

(STBASST) Our Price **$89**



JavaScript For The World Wide Web

by Ted Gesing and Jeremy Schneider

This book takes an easy, visual approach to teaching JavaScript, where pictures guide you through the software and show you what to do. Works like a reference book, you look up what you need and then get straight to work. No long winding passages, concise, straightforward commentary explains what you need to know.

(BJWWW) Our Price **$16.15**

Metrowerks CodeWarrior Programming

by Dan Parks Sydow

Includes CodeWarrior Lite, and Full Coverage of PowerPlant™. The best information on Metrowerks CodeWarrior, giving full coverage to the Gold Edition. CD includes Code Warrior Lite.

(BCWPROG) Our Price **$35**

CodeWarrior Software Development Using PowerPlant

by Jan L. Harrington

C++ programmers will learn to develop object-oriented software applications for the Mac and Power Mac using the PowerPlant environment and the classes that support it. Covers CodeWarrior 8. Included CD-ROM contains source code for all the programming examples in the book and Metrowerks CodeWarrior Lite.

(BCWSWDEV) Our Price **$31**

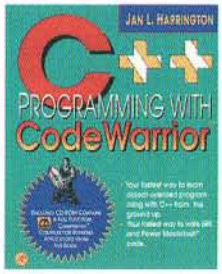
Inside PowerPlant

by Metrowerks

Create PowerPlant applications using the CodeWarrior IDE and PowerPlant Constructor. Full descriptions of major PowerPlant classes and resources. Included are the PowerPlant Constructor Manual, including View, TextTraits and Custom Types editing, and PowerPlant Library Reference, covering all classes and functions in PowerPlant.

(BINSPP) Our Price **$34**

SEE RELATED CATEGORY: Dev. Environment

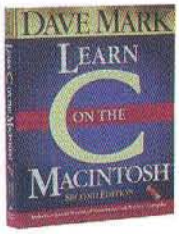


C++ Programming with CodeWarrior

by Jan L. Harrington

Beginning OOP for the Macintosh and Power Macintosh and Mac OS compatibles. Learn object-oriented programming techniques using C++ as the example language and Metrowerks and CodeWarrior as the example compiler. Enclosed CD contains example code from the book and a full-function Metrowerks CodeWarrior.

(BCPPCW) Our Price **$33**



Learn C on The Macintosh, Second Edition

by Dave Mark

New revised edition! Easy-to-understand – everything you need to start programming. Updated and enhanced exercises that lead you step by step. You'll learn function, variables, point datatypes, data structures, file input and output and more! Includes CD-ROM with Metrowerks CodeWarrior™ Lite.

(BLEARNC2) Our Price **$33**

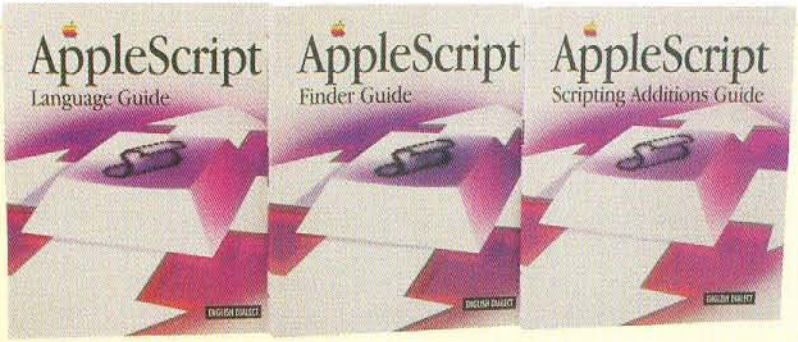
Order Toll-free
800-MACDEV-1
(800-622-3381)

MacTech® Magazine

MacTech keeps Mac programmers & developers up to date with everything they need to know about software development. Topics like Rhapsody, Java, QuickTime, OPENSTEP, Objective-C, C/C++, Object Oriented Technologies, product reviews and much more!

Subscriptions:

(MTYRDM) US/Domestic for 12 issues **$47**
(MTYRCM) Canadian for 12 issues **$59**
(MTYRFM) International for 12 issues **$97**
Back Issues: each plus shipping (subject to availability) **$10**



AppleScript Language Guide

by Apple Computer, Inc.

A complete reference for anyone using AppleScript to modify existing scripts or to write new ones. Contains useful information for programmers who are working on scriptable applications or complex scripts. Features detailed definitions of AppleScript terminology and syntax in the following categories: Value classes, commands, objects and references to objects, expressions, control statements, handlers, and script objects. Includes many sample scripts, discusses advanced topics such as writing command handlers for script applications, the scope of script variables and properties declared at different levels in a script, and inheritance and delegation among script objects.

(BALG) Our Price **$26.95**

SEE RELATED CATEGORY: Scripting

AppleScript Applications: Building Applications with FaceSpan and AppleScript

by John Schettino Affiliation & Liz O'Hara

Build complete AppleScript applications using FaceSpan, a user interface development tool that makes AppleScript applications truly "Mac-Like". Uses a step-by-step approach to demonstrate techniques for building applications through illustrations and samples. Provides Graphical User Interface (GUI) design tips and practical approaches for implementation. Contains one CD-Rom with AppleScript 1.1, a demonstrations version of FaceSpan 2.1, source code for all example applications numerous AppleScript shareware and demonstrations programs. Contains a section on debugging AppleScript applications using FaceSpan.

(BAPSCAP) Our Price **$31**

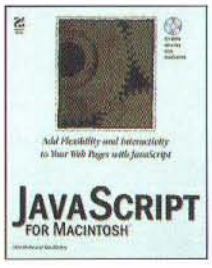
Special Edition Using CGI, 2nd Edition

by Jeffry Dwight, Michael Erwin and Robert Niles

This complete reference provides professional Web developers and advanced personal users with the latest information on using CGI (Common Gateway Interface) to interact with databases.

- Explains client and server uses of CGI
- Provides extensive coverage of live audio and video feeds, user chat and interaction, and CGI security
- Features separate chapters devoted to language-specific tips, tricks, and traps
- CD ROM is loaded with the HTML and CGI sample code from the book
- Includes applications for guest books, mail and new gateways, browser identification, access restriction, and shopping carts

(BSEUCGI) Our Price **$44**

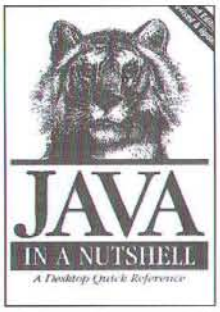


JavaScript for the Macintosh

by Matt Shobe and Tim Ritchey

Allows non-programmers to take advantage of the power of Netscape Navigator. Expand the capabilities of your Web page, without having to understand C or C++. CD-ROM contains "Wizlets" that allows you to easily create your own JavaScripts. Takes you step-by-step through programming cross-platform JavaScripts. Details how to create JavaScripts for JavaScript-aware Web browsers.

(BJAVASCRPTJ) Our Price **$40**



Java in a Nutshell, 2nd Edition

by David Flanagan

A detailed overview of all of the new features in Java 1.1, both on a package-by-package basis and in terms of overall functionality. A comprehensive tutorial on "inner classes" that explains how to use all of the new types of inner classes: static member classes, member classes, local classes, and anonymous classes. Practical, real-world example programs that demonstrate the new features in Java 1.1, including object serialization, the new AWT event handling model, internationalization, and a sample Java Bean.

(BJNUT2) Our Price **$17.95**

AppleScript Finder Guide, English Dialect

by Apple Computer, Inc.

Provides definitions for Finder object classes and commands. Write, record, or run scripts that trigger the same desktop actions that you trigger using the keyboard and mouse.

(BAFG) Our Price **$17.95**

SEE RELATED CATEGORY: Scripting

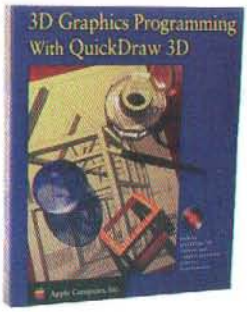
Inside CodeWarrior Professional

by Metrowerks

Includes CodeWarrior IDE User's Guide. This is the printed version of the documentation provided on the CD. Covers CodeWarrior Professional Release, the debugger and associated tools.

(BINSCWP) Our Price **$34**

SEE RELATED CATEGORY: Dev. Environment



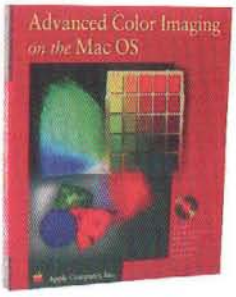
3D Graphics Programming Using QuickDraw 3D

Includes CD-ROM!

by Apple Computer, Inc.

Incorporate spectacular 3D graphics into your applications. Explore QuickDraw 3D, a revolutionary graphics extension to the Mac OS for Power Macintoshes. CD contains the complete QuickDraw 3D system itself and a complete database of the QuickDraw 3D API, allowing you instant access to the hundreds of graphics calls via a fast viewing engine. Book/CD-ROM, 640 pages.

(B3DGRAP) Our Price **$35**



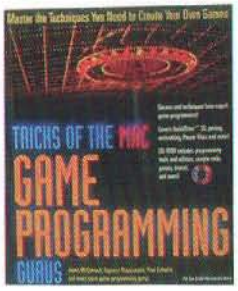
Advanced Color Imaging on the Mac OS

Includes CD-ROM!

by Apple Computer, Inc.

Enhance your software's color capabilities with step-by-step instructions. Augment the color support supplied with QuickDraw, and QuickDraw GX. Use the Pallette Manager to get the best colors on limited displays. Match colors between screens and input/output devices (scanners & printers). CD includes a complete reference information in both QuickView and Acrobat formats. Plus, a sample application demonstrating ColorSync programming techniques.

(BADVCI) Our Price **$33**

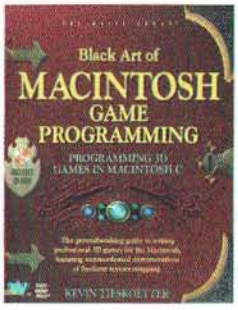


Tricks of The Mac Game Programming Gurus

by McCornack, Ragnemalm, Celestin, et al.

For beginning to expert game programmers. Complete overview of all the necessary components of game programming on the Macintosh. Packed with valuable tools, utilities, sample code, CodeWarrior™ Lite and game demos. QuickDraw 3D and Power Mac optimization and inside info on how Glypha III was created. Hundreds of tried-and-true tricks, tips, and insider secrets from well-known Mac game programming experts.

(BTRICKS) Our Price **$45**



Black Art of Macintosh Game Programming

Includes CD-ROM!

by Kevin Tieskoetter

Develop your own 3D games in C on the Mac. Includes CD with project files for both Symantec C and Code Warrior. Create freeform texture-mapped games and polygon graphics. Control dynamic source code — all compatible as native to the Power Mac. Write directly to the screen, bypassing QuickDraw.

(BBLACK) Our Price **$35**

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us. Our prices on books are at least 10% off list price.

PRODUCT	CODE	PRICE
Development Environments		
AppleScript Applications: Building Applications w/FaceSpan	BAPSCAP	$31.00
C++ Programming	BCPPMACAP	31.00
C/C++ SDK User's Guide	BCPPUSER	29.00
CodeWarrior Inside PowerPlant	BINSPP	34.00
Cyberdog Programmers Kit	BCYBERDOG	34.00
Dan Shafer Presents the Power of Prograph CPX	BDANPRO	19.95
Inside CodeWarrior Book	BINSCW	34.00
Instant CORBA	BIC	17.99
Last Resort Programmers Edition	BLSTRSRT	74.00
Macintosh C Programming Primer Volume I	BCPRIM1	24.25
Macintosh C Programming Primer Volume II	BCPRIM2	24.25
Macintosh Pascal Programming Primer Volume I	BPASCPRI	24.25
Mastering the Think Class Library	BMASTERTCL	26.95
Mastering the Toolbox Using THINK C	BCPRIM2	24.25
Objective-C - Object Oriented Programming	BOBJCOOPT	34.00
Presenting Magic Cap	BPRESMAGIC	15.25
Real World Apple Guide	BREALWLD	35.00

PRODUCT	CODE	PRICE
Symantec C++ Programming	BSYMCPP	39.00
Taligent's Guide to Designing Programs	BTALIGENT	17.55
The Power of Prograph CPX	BDANPRO	19.95
Visual Programming with Prograph CPX	BVISPRO	30.00
Wireless For The Newton	BWIRELESS	31.00
Hardware		
Apple CD-ROM Handbook	BCDHAND	14.36
Designing Cards & Drivers for the Macintosh	BCARD	26.96
LaserWriter Reference	BLASERREF	17.96
PCI System Architecture 3rd Edition	BPCISYS	31.00
PowerPC System Architecture	BPPCARCH	31.00
Internet Related		
1994 Internet White Pages	BHWHITE	26.95
Active Java	BACTJAVA	23.36

America Online for Dummies	BAOLDUM	17.95
CGI By Example	BCGIBE	31.00
Building & Maintaining an Intranet w/ the Macintosh	BBAMAI	45.00
Claris Home Page Companion	BCHPC	26.95
Computer Privacy Handbook	BPRIV	22.45
Developing CGI Applications with Perl	BDCGIA	26.95
E-Mail Essentials	BEMAILE	22.45
Elements of E-Mail Style	BEMAIL	13.45
Hooked on Java	BHJAVA	26.95
Instant Internet Guide	BINSTANT	13.45
Internet Book	BTHENET	22.50
Internet for Dummies 2nd Edition	BNETDUM2	17.99
Internet for Dummies Quick Reference	BDUMQCK	8.05
Internet for Macs for Dummies	BNETDUM	17.95
Internet for Macs for Dummies Bestseller Edition	BFMFDBE	35.00
Internet Power Tools	BPWRTOOL	36.00
Internet Publishing with Adobe Acrobat	BIPWAA	36.00
Internet, The, Deluxe Edition	BNETDELUX	31.00
Intranet Web Dev.: Enterprise Alternatives to Client/Server	BINTWD	44.00
Java Essentials for C/C++ Programmers	BJAVAESSEN	17.95
Java in a Nutshell	BJAVANUT	13.45
Java Language API SuperBible	BJLAS	53.00
Java Programming with CORBA	BJPWC	26.99
JavaScript Cookbook	BJSCB	44.00
JavaScript for Macintosh	BJAVASCRPT	40.00
Learn HTML on the Macintosh	BLHTML	26.95
Learn Java on the Macintosh	BLJAVA	31.00
Mastering Netscape 2.0 for Macintosh, Second Edition	BMASNET2	36.00
Measuring the Impact of Your Web Site	BMIYWS	26.95
More Internet for Dummies Starter Kit	BDUMNET	17.95
Mosaic for Dummies	BMOSDUM	17.99
Net Chat	BNETCHAT	17.00
NetObjects Fusion Handbook	BNETOFH	45.00
Perl Quick Reference	BPERLREF	17.99
Planning and Managing Websites	BPLANWEB	35.00
Protect Your Privacy on the Internet	BPYP	26.99
Providing Internet Services via the MacOS	BPROVNET	31.00
Publish it on the Web	BWEBPUB	31.00
TCP/IP Vol 1-Vol 2 Bundle	BTCP12BNDL	99.00
Teach Yourself Java in 21 days	BJAVAMAC	36.00
The Internet Marketing Plan	BTIMP	35.00
The Internet Strategic Plan	BTISP	22.45
The Web Navigator	BTWN	22.45
Underground Guide to Telecommuting	BUNDER	22.45
Using Lotus Notes as an Intranet	BULN	40.00
Web Graphics Tools and Techniques	BWGTT	35.00
Web Head Mac Guide	BWEBHEAD	22.45
Web Marketing Cookbook	BWMCB	35.00
Web Page Scripting Techniques	BWEBPST	45.00
Web Publishing with Adobe Acrobat and PDF	BWPWAA	35.95
Web Weaving	BWWEAV	22.45
Webmaster Macintosh	BWEBMAS	26.95

Scripting and Solutions

AppleScript Applications: Building Apps with FaceSpan	BAPSCAP	31.00
AppleScript Scripting Additions Guide	BSCRADD	17.05
Applied Macintosh Scripting	BAPPLIED	31.00
Complete HyperCard 2.2 Handbook	BHYPCRD2	31.00
Complete HyperTalk 2.2:	BHYPCRD2	31.00
Danny Goodman's Apple Guide Starter Kit	BDGAGSK	31.00
HyperCard Stack Design	BHYPSTA	19.95
JavaScript for Macintosh	BJAVASCRPT	40.00
Perl Quick Reference	BPERLREF	17.99
Real World Apple Guide	BREALWLD	35.00

Technical Reference

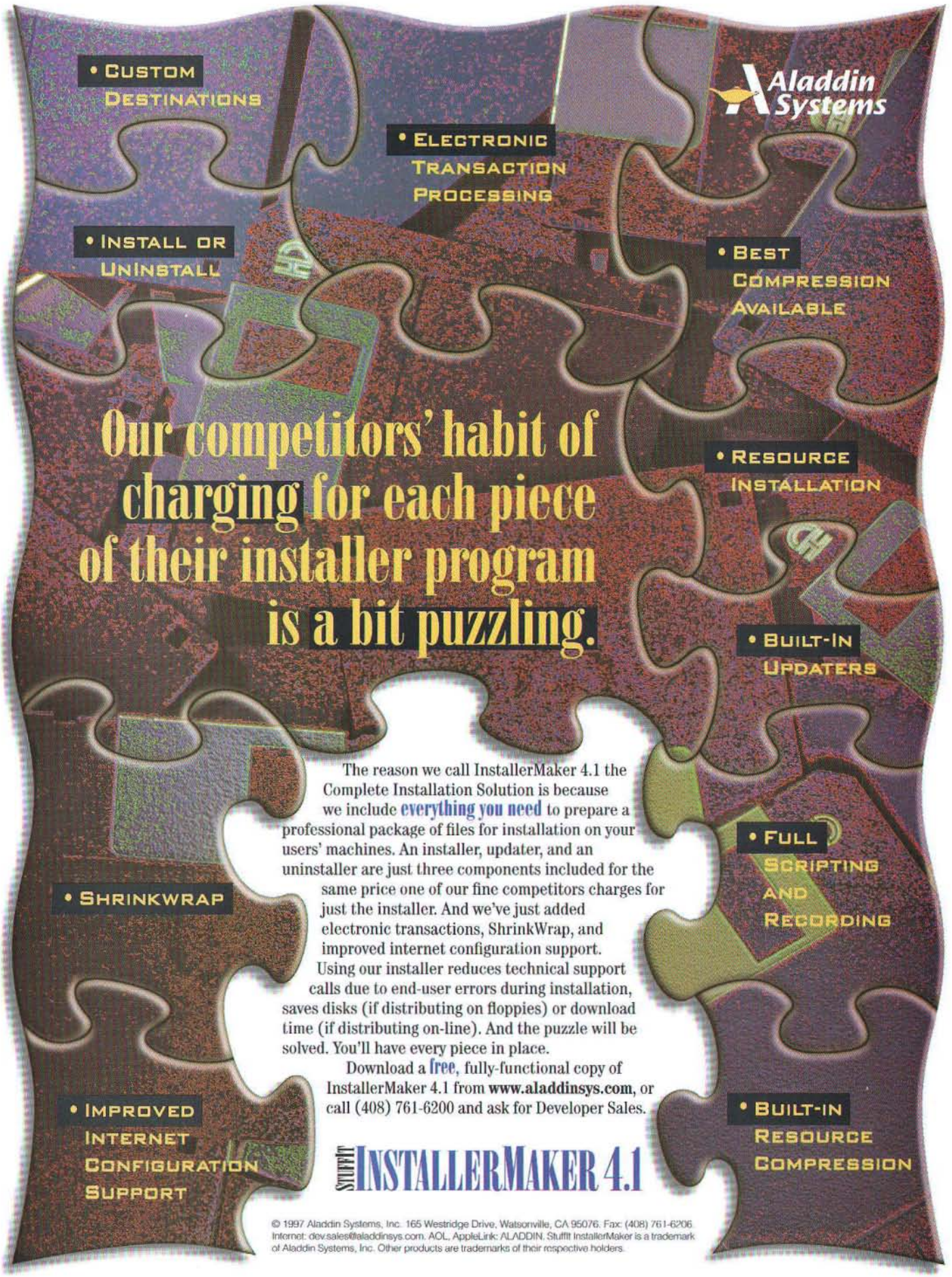
Active Java	BACTJAVA	23.36
Apple CD-ROM Handbook	BCDHAND	14.36
Art of Human Interface Design	BAHID	29.65
Black Art of Mac Game Programming	BBLACK	35.00
C++ for Dummies	BCPPDUM	22.46
C for Dummies Vol. 1	BCDUM	17.95
Developing Object Oriented Software for the Macintosh	BDEVOB	26.06
Extending the Mac Toolbox	BETMT	22.46
Foundations of Macintosh Programming	BFOUND	35.00
Fragment of Your Imagination	BFRAG	35.00
Guide to Macintosh Software Localization	BLOCALIZ	24.26
Guide to Macintosh System 7.5	BSYS7.5	22.50
Inside AppleTalk	BAPTALK	31.00
Inside the Macintosh Communications Toolbox	BCOMM	22.45
LaserWriter Reference	BLASERREF	17.96
Learn C++ on the Macintosh	BLRNCPP	35.00
Learn C on the Macintosh, 1st Edition	BLEARNC1	31.00
Learn C on the Macintosh, 2nd Edition	BLEARNC2	33.00
Mac Programming for Dummies	BMACDUM	17.95
Macintosh C Programmer Primer Volume 1	BCPRIM1	24.25
Macintosh C Programmer Primer Volume 2	BCPRIM2	24.25
Macintosh OLE2 Prog. Reference Working with Objects	BOLE2	40.00
Macintosh Pascal Programming Primer Volume I	BPASCPRI	24.25
Macintosh Programming Secrets 2nd edition	BPSECRET	28.76
Macintosh Programming Techniques	BPTECH	31.00
Microsoft Visual J++ 1.1 Sourcebook	BMVJS	35.00
More Mac Programming Techniques	BMORETECH	34.00
Network Frontiers Bundle	BNETFB	59.00
Newton Programming Guide	BNEWTPGUID	40.00
Object Oriented Programming Design	BOOPRODES	20.66
Optimizing PowerPC Code	BOPTPPC	35.00
Perl Quick Reference	BPERLREF	17.99
PostScript Language Reference	BPSLANREF	29.66
Powerbook: Digital Nomad's Guide	BPBTDNG	22.46
PowerPC Programmer's Toolkit	SPPCPT	40.00
Programming Introduction to the Macintosh Family	BFAMILY	22.46
Programming for System 7	BSYS7	24.25
Programming with AppleTalk	BPROAT	22.45
QuickTime - Official Guide for Macintosh Users	BQTGUIDE	45.00
Real World Apple Guide	BREALWLD	35.00
ResEdit All Night Diner	BRESDINE	22.45
ResEdit Complete, 2nd Edition	BRESED2	31.00
ResEdit Reference	BRESEDREF	26.96
Software by Design: Creating User Friendly Software	BDESIGN	26.95
Symantec C++ for the Macintosh: The Basics	BSCFTMTB	31.00
Teach Yourself Macintosh C++ in 21 Days	BCPP21D	26.99
Technical Introduction to the Macintosh Family	BTITTMF	24.26
Tog on Software Design	BTOG	26.95
Writing Localizable Software	BLOCAL	24.25

Miscellaneous

3D Graphics-Tips, Tricks, & Techniques	B3DGTTT	31.00
A Fragment of Your Imagination	BFRAG	35.00
Adobe Premiere for the Macintosh	BPREM	44.00
America Online for Dummies	BAOLDUM	17.95
AppleGuide Complete	BAPLGD	35.00
Art of Human Interface Design	BAHID	35.00
CD-ROM Guide to Multimedia Authoring	BCDMULTI	40.00
CompuServe for Dummies	BCSDUM	17.95
Cyberpunk Handbook	BCYBPUNK	8.95
Danny Goodman's Apple Guide Starter Kit	BDGAGSK	31.00
Danny Goodman's Macintosh Handbook	BGOODHB	26.95
FrameWorks Source Code Disk	MTFWSC	9.95
FrameWorks Magazine Back Issue	MTFWBACK	8.00
Graphic Gems 2	BGEMS2	44.00
Graphic Gems 4	BGEMS4	44.00
Graphic Gems V	BGEMS5	44.00
Infini-D Revealed	BINFDREV	40.00
Inside Director 5 with Lingo for Macintosh	BID5WLM	44.00
Late Night with MacHack	BLATE	26.95
Mac Bathroom Reader	BBATH	11.70
Macromedia Director Lingo Workshop, 2nd Edition	BMDLW2	40.00
Mac Screamer: The Ultimate Macintosh Supercharging Kit	BSCRFAM	31.00
Macintosh Crash Course	BCRASH	26.95
MacTech Back Issues	MTBACKISS	10.00
Macworld Ultimate Macintosh Programming Book	BULTMAC	35.00
MADACON '93 CD-ROM	SMADA93	9.95
Multimedia Authoring: Building and Developing Documents	BMMAUTH	31.00
Multimedia Starter Kit for Macintosh	BMMSTART	27.00
Profit from Experience	BPROFIT	22.45
ResEdit Complete, Second Edition	BRESED2	31.00
Sad Macs, Bombs and Disasters	BSADMAC	22.45
Standards For Online Communication	BSFOC	40.00
The Elements of E-Mail Style	BEMAIL	13.45
The Software Developer's & Marketer's Legal Companion	BSDAMLC	33.00
Tog on Software Design	BTOG	26.95
Tricks of the Mac Game Gurus	BTRICKS	45.00
Zen and the Art of Resource Editing	BZAAORE	27.00

All entries in this index are alphabetized. For your convenience the product names are **bold**, book names are *italicized* and company names are in plain text.

INDEX


• Custom Destinations
• Electronic Transaction Processing
Aladdin Systems
• Install or UnInstall
• Best Compression Available
Our competitors' habit of charging for each piece of their installer program is a bit puzzling.
• Resource Installation
• Built-In Updaters
The reason we call InstallerMaker 4.1 the Complete Installation Solution is because we include everything you need to prepare a professional package of files for installation on your users' machines. An installer, updater, and an uninstaller are just three components included for the same price one of our fine competitors charges for just the installer. And we've just added electronic transactions, ShrinkWrap, and improved internet configuration support. Using our installer reduces technical support calls due to end-user errors during installation, saves disks (if distributing on floppies) or download time (if distributing on-line). And the puzzle will be solved. You'll have every piece in place.
Download a free, fully-functional copy of InstallerMaker 4.1 from www.aladdinsys.com, or call (408) 761-6200 and ask for Developer Sales.
• Full Scripting and Recording
• ShrinkWrap
• Improved Internet Configuration Support
• Built-In Resource Compression
StuffIt InstallerMaker 4.1
© 1997 Aladdin Systems, Inc. 165 Westridge Drive, Watsonville, CA 95076. Fax: (408) 761-6206. Internet: dev.sales@aladdinsys.com. AOL, AppleLink: ALADDIN. StuffIt InstallerMaker is a trademark of Aladdin Systems, Inc. Other products are trademarks of their respective holders.

CODEWARRIOR LATITUDE

KICKING BUTT AND WRITING CODE

CodeWarrior Latitude

When Rhapsody gets here, you'll be ready...with CodeWarrior Latitude™. It's Metrowerks' newest porting tool, designed to give you a leg up on Rhapsody. Recompile your Mac OS source code, link it with the Latitude libraries and find out which portions of your code are going to port smoothly...and which won't [forewarned is forearmed]. As Rhapsody evolves, so will Latitude; registered users will receive all developer releases, the first full release, plus one additional update. CodeWarrior Latitude. $399. The tools you need...and a little attitude to boot.

COOL TOOLS FOR KILLER CODE.

metrowerks®

http://www.metrowerks.com